Intel® oneAPI Math Kernel Library LAPACK Examples

Contents

Chapter 1: Intel® oneAPI Math Kernel Library LAPACK Examples

Legal Information 3
Intel® oneAPI Math Kernel Library LAPACK Examples 4
What's New 4
Notational Conventions 5
C Interface Conventions 5
Compiling and Running Examples 7
Compiling and Building Fortran Examples 8
Compiling and Building C Examples 9
Getting Help and Support 10
LAPACK Examples 11
LAPACK Routines: Linear Equations 12
?gesv function 13
CGESV Example 14
DGESV Example 25
SGESV Example 37
ZGESV Example 48
?posv function 59
CPOSV Example 60
DPOSV Example 70
SPOSV Example 80
ZPOSV Example 90
?sysv function 100
CSYSV Example 101
DSYSV Example 112
SSYSV Example 124
ZSYSV Example 136
?hesv function 148
CHESV Example 148
ZHESV Example 160
Least Squares and Eigenvalue Problems 172
Linear Least Squares (LLS) Problems 172
?gels function 173
?gelsd function 217
Symmetric Eigenproblems 262
?syev function 263
?heev function 281
?syevd function 301
?heevd function 320
?syevx function 341
?heevx function 361
?syevr function 384
?heevr function 405
Nonsymmetric Eigenproblems 429
?geev function 429
Singular Value Decomposition 474
?gesvd function 475
?gesdd function 519

Intel® oneAPI Math Kernel Library LAPACK Examples

1

This document provides code examples for LAPACK (Linear Algebra PACKage) routines that solve problems in the following fields:

Linear Equations

Examples for several LAPACK routines that solve systems of linear equations.

Linear Least Squares Problems

Examples for some of the LAPACK routines that find solutions to linear least squares problems.

Symmetric Eigenproblems

Symmetric Eigenproblems has examples for LAPACK routines that compute eigenvalues and eigenvectors of real symmetric and complex Hermitian matrices.

Nonsymmetric Eigenproblems

Nonsymmetic Eigenproblems provides examples for *?geev*, one of several LAPACK routines that compute eigenvalues and eigenvectors of general matrices.

Singular Value Decomposition

Examples for LAPACK routines that compute the singular value decomposition of a general rectangular matrix.

> **NOTE**
> If you want to find a routine for a *specific purpose*, use the oneMKL LAPACK Function Finding Advisor. It advises you which routine to use according to parameters that you specify.

Legal Information

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may

cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Portions© Copyright 2001 Hewlett-Packard Development Company, L.P.

Sections on the Linear Algebra PACKage (LAPACK) routines include derivative work portions that have been copyrighted:

© 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.

Intel® oneAPI Math Kernel Library LAPACK Examples

LAPACK - Linear Algebra PACKage, is a de-facto standard of the linear algebra functionality. The reference code is provided at Netlib site http://www.netlib.org/lapack.

Intel® oneAPI Math Kernel Library provides multi-core optimized high performance LAPACK functionality consistent with the standard.

The Intel® oneAPI Math Kernel Library (oneMKL) LAPACK examples are Fortran and C source files that illustrate how to call LAPACK routines in the oneMKL library.

The examples show how to:

- fill the input matrices in column-major order
- pass the parameters
- query the optimal size of the work arrays
- interpret the output results

For more information on LAPACK functionality, please refer to Intel® oneAPI Math Kernel Library Developer Reference at https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html or to the Netlib site at http://www.netlib.org/lapack.

Follow the links to the general LAPACK naming conventions and information on how to obtain the required example:

- Notational Conventions
- Compiling and Running Examples

Follow the links below for information on how to compile and build examples with different bindings:

- Compiling and Building Fortran Examples
- Compiling and Building C Examples

This document includes the following sections:

- Routines and Examples index
- Linear Equations
- Least Squares and Eigenvalue Problems

What's New

This online document provides Intel® oneAPI Math Kernel Library (oneMKL) LAPACK code examples.

In the current version, minor improvements have been made to reflect changes in Intel® oneAPI Math Kernel Library Developer Reference. The document title page has been improved. Compiling and Building Examples sections for both C and Fortran have been reworked. Additionally, changes have been made to correct errors in the document.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Product and Performance Information
Notice revision #20201201

Notational Conventions

This document uses the following conventions:

Convention	Description
`lowercase courier`	Indicates routine names.
`lowercase courier italic`	Indicates routine arguments.

The LAPACK routines are prefixed by a letter defining the data type:

Letter	Description
s	Indicates single precision: REAL, or REAL*4 in Fortran, float in C.
d	Indicates double precision: DOUBLE PRECISION, or REAL*8 in Fortran, double in C.
c	Indicates single precision complex: COMPLEX, or COMPLEX*8 in Fortran, single precision complex data type in C. See the note about complex data type below.
z	Indicates double precision complex: DOUBLE COMPLEX, or COMPLEX*16 in Fortran, double precision complex data type in C. See the note about complex data type below.
i	Indicates integer: INTEGER in Fortran, int in C.
?	Indicates a place holder for a single letter in several routine names. For example, `?getrf` refers to the routines `sgetrf`, `dgetrf`, `cgetrf`, and `zgetrf`.

Note: Complex data type in C means a type containing two consecutive float or double fields representing real and imaginary parts of the complex number. Complex types `fcomplex` and `dcomplex` in oneMKL LAPACK examples are defined as follows:

```
/* Single precision complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;
/* Double precision complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;
```

C Interface Conventions

The C interfaces are implemented for most of the oneMKL LAPACK driver and computational routines.

The arguments of the C interfaces for the oneMKL LAPACK functions comply with the following rules:

- Scalar input arguments are passed by value.
- Array arguments are passed by reference.
- Array input arguments are declared with the `const` modifier.
- Function arguments are passed by pointer.
- An integer return value replaces the *`info`* output parameter. The return value equal to 0 means the function operation is completed successfully. See the section Error Codes below.
- Most of the LAPACK C interfaces have an additional parameter *`matrix_layout`* of the `int` type as their first argument. This parameter specifies whether the two-dimensional arrays are row-major (`LAPACK_ROW_MAJOR`) or column-major (`LAPACK_COL_MAJOR`).

The LAPACK C interface omits workspace parameters because workspace is allocated at run time and released upon completion of the function operation.

For the ?gesvd function, *work* arrays contain valuable information on exit. The interface contains an additional argument *superb*.

Function Types

The function types are used in non-symmetric eigenproblem functions only.

```
typedef lapack_logical (*LAPACK_S_SELECT2) (const float*, const float*);
typedef lapack_logical (*LAPACK_S_SELECT3) (const float*, const float*, const float*);
typedef lapack_logical (*LAPACK_D_SELECT2) (const double*, const double*);
typedef lapack_logical (*LAPACK_D_SELECT3) (const double*, const double*, const double*);
typedef lapack_logical (*LAPACK_C_SELECT1) (const lapack_complex_float*);
typedef lapack_logical (*LAPACK_C_SELECT2) (const lapack_complex_float*, const
lapack_complex_float*);
typedef lapack_logical (*LAPACK_Z_SELECT1) (const lapack_complex_double*);
typedef lapack_logical (*LAPACK_Z_SELECT2) (const lapack_complex_double*, const
lapack_complex_double*);
```

Mapping FORTRAN Data Types against C Data Types

FORTRAN Data Types	C Data Types
INTEGER	lapack_int
LOGICAL	lapack_logical
REAL	float
DOUBLE PRECISION	double
COMPLEX	lapack_complex_float
COMPLEX*16	lapack_complex_double
CHARACTER	char

C Types

```
#ifndef lapack_int
#define lapack_int MKL_INT
#endif
#ifndef lapack_logical
#define lapack_logical lapack_int
#endif
```

Complex Types

For single precision:

```
#ifndef lapack_complex_float
#define lapack_complex_float   MKL_Complex8
#endif
```

For double precision:

```
#ifndef lapack_complex_double
#define lapack_complex_double   MKL_Complex16
#endif
```

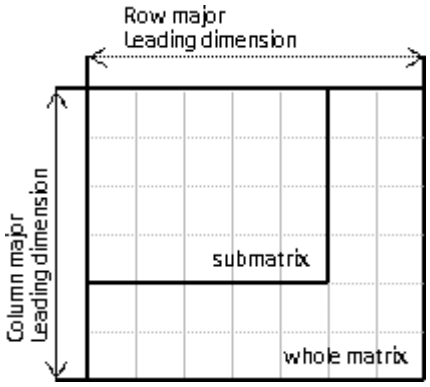
Matrix Layout

Most of the LAPACK C interfaces have an additional parameter matrix_layout of type int as their first argument. This parameter specifies whether the two-dimensional arrays are row-major LAPACK_ROW_MAJOR or column-major LAPACK_COL_MAJOR.

In general the leading dimension `lda` is equal to the number of elements in the major dimension. It is also equal to the distance in elements between two neighboring elements in a line in the minor dimension. If there are no extra elements in a matrix with `m` rows and `n` columns, then

- For row-major ordering: the number of elements in a row is `n`, and row `i` is stored in memory right after row `i`-1. Therefore `lda` is `n`.
- For column-major ordering: the number of elements in a column is `m`, and column `i` is stored in memory right after column `i`-1. Therefore `lda` is `m`.

To refer to a submatrix with dimensions `k` by `l`, use the number of elements in the major dimension of the whole matrix (as above) as the leading dimension and `k` and `l` in the subroutine's input parameters to describe the size of the submatrix.



Matrix Ordering

```
#define LAPACK_ROW_MAJOR  101
#define LAPACK_COL_MAJOR  102
```

Error Codes

```
#define LAPACK_WORK_MEMORY_ERROR       -1010  /* Failed to allocate memory
                                                 for a working array */
#define LAPACK_TRANSPOSE_MEMORY_ERROR  -1011  /* Failed to allocate memory
                                                 for transposed matrix */
```

If the return value is `-i`, the `-i`-th parameter has an invalid value.

Function Prototypes

Some oneMKL functions differ in data types they support and vary in the parameters they take.

Each function type has a unique prototype defined. Use this prototype when you call the function from your application program. In most cases, oneMKL supports four distinct floating-point precisions. Each corresponding prototype looks similar, usually differing only in the data type. To avoid listing all the prototypes in every supported precision, a generic prototype template is provided. For example,

```
lapack_int LAPACKE_
```

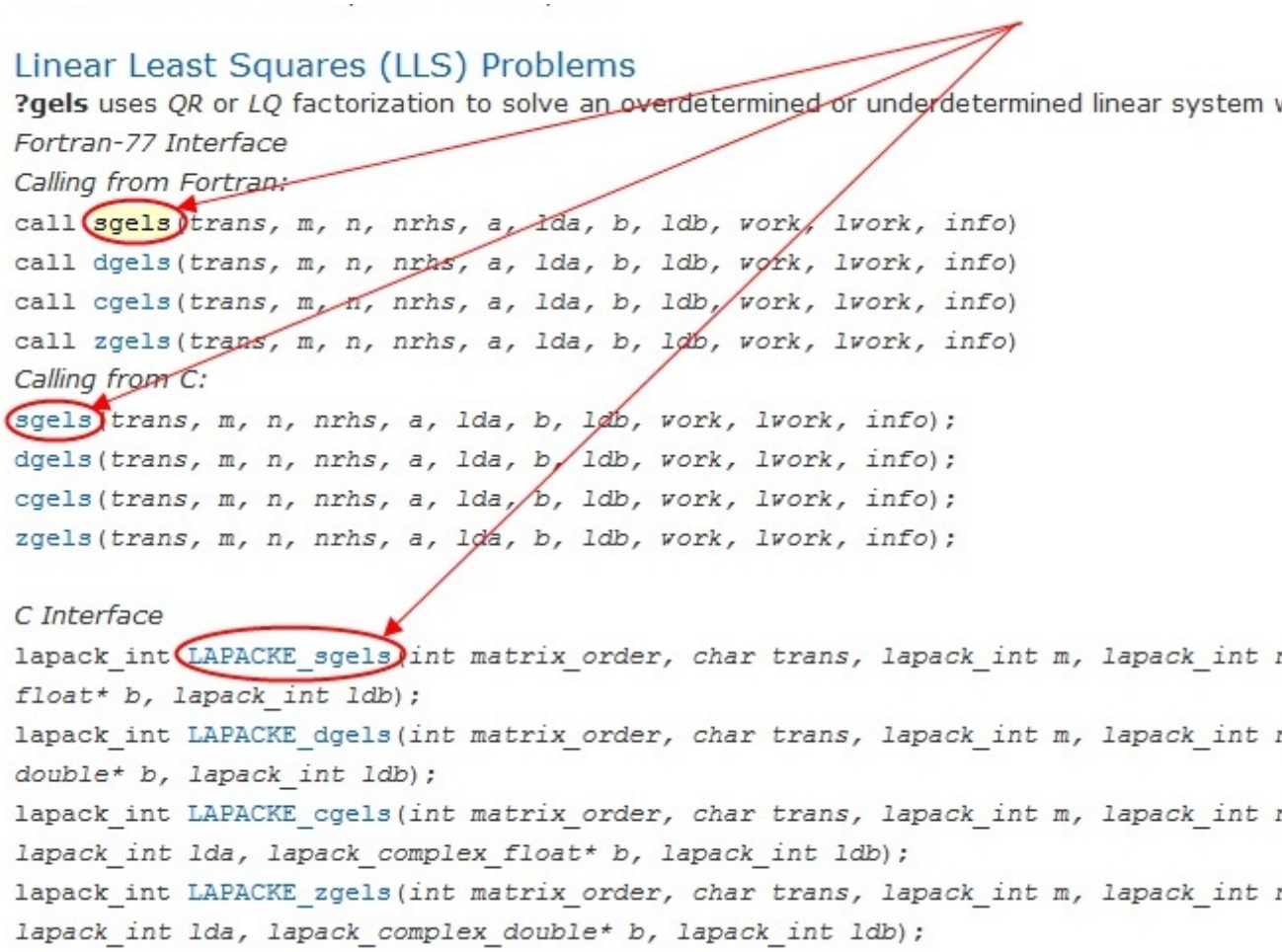
Compiling and Running Examples

The examples contain sample input data in their body.

C and Fortran examples for the same routine contain exactly the same input data and should produce the same output, with possible differences in the amount of white spaces. For -0.00 and 0.00 (signed and unsigned zero), you can safely ignore these differences. The same input data is provided for double and

single precision examples, for example, `dgels` and `sgels`, or `cgeev` and `zgeev`. The output of the same routine in different precisions should differ only after several digits in mantissa, because single precision routines provide less precise results than double precision routines.

1. Open an example topic by clicking the respective routine name in any list of routines in the API section.



2. Compile and link with your LAPACK software (oneMKL or any other LAPACK implementation), using a related compiler. For example, use a C compiler for the C examples. You can make use of the web-based Intel® oneAPI Math Kernel Library Link Line Advisor to create a correct link line.
3. Run the compiled example.

See Also

Compiling and Building Fortran Examples
Compiling and Building C Examples

Compiling and Building Fortran Examples

If possible, compile the Fortran examples with the Fortran 90 compiler, because some examples contain Fortran 90 specific features like the `ADVANCE` specifier in the `WRITE` statement.

An LAPACK binary defines the appropriate entry point for the LAPACK routine, which is called from the example according to the default Fortran naming convention. For example, to define the `dgesv` routine on IA-32 architecture, use `dgesv_` (lowercase, trailing underscore) on Linux* OS, or `_DGESV` (uppercase, leading underscore) on Windows* OS.

To compile and build an example:

1. Run the `mklvars` script to set the required environment variables.

2. Build the application using a Fortran 90 compiler as follows:

   <f90> <compiler options> <example.f> <link line>

   where *<f90>* is a Fortran 90 compiler, *<example.f>* is any of the oneMKL LAPACK Fortran examples, and *<link line>* contains linker options and arguments, including the LAPACK library files required for your application.

   See the following example lines for reference:

   - with Intel® Fortran Compiler, you can use

     `ifort -mkl cgesv_ex.f` on Linux* OS, or

     `ifort /Qmkl cgesv_ex.f` on Windows* OS.

     See the *Linking Your Application with the Intel oneAPI Math Kernel Library* section of the oneMKL Developer Guide for more information on linking with the oneMKL.
   - with third-party compilers, you can use

     `gfortran -fopenmp -m64 -I${MKLROOT}/include cgesv_ex.f -Wl,--no-as-needed -L${MKLROOT}/lib/intel64 -lmkl_gf_lp64 -lmkl_core -lmkl_gnu_thread -lpthread -lm -ldl` to compile with the GNU* Fortran compiler on Linux* OS.

NOTE The example above is linking dynamically against 64-bit oneMKL library files. To create a custom link line, use the oneMKL Link Line Advisor or refer to the oneMKL Developer Guide, section *Linking Your Application with the Intel oneAPI Math Kernel Library*.

See Also

oneMKL Developer Guide Linux* OS section *Linking Your Application with the Intel oneAPI Math Kernel Library*
section *Linking Your Application with the Intel oneAPI Math Kernel Library*
oneMKL Developer Guide Windows* OS section *Linking Your Application with the Intel oneAPI Math Kernel Library*
section *Linking Your Application with the Intel oneAPI Math Kernel Library*
oneMKL Developer Guide Linux* OS section *Getting Started > Setting Environment Variables*
section *Getting Started > Setting Environment Variables*
oneMKL Developer Guide Windows* OS section *Getting Started > Setting Environment Variables*
section *Getting Started > Setting Environment Variables*
oneMKL Link Line Advisor

Compiling and Building C Examples

When using an ANSI C compiler, the C example calls the LAPACK routine in lowercase with no underscore:

```
dgesv
```

oneMKL defines the `dgesv` routine using the standard Fortran-style: `dgesv_` (lowercase, trailing underscore) on Linux* OS, or `_DGESV` (uppercase, leading underscore) on Windows* OS. Another LAPACK binary, such as the one compiled from Netlib Fortran reference code, may define only the Fortran entry points. If you want to link a C example against such a binary, you need to redefine the C routine name to avoid an unresolved external reference, for example:

- On Linux* OS:

```
<cc> <compiler options>
     -Ddgesv=dgesv_ dgesv_ex.c <link line>
```

```
<cc> <compiler options>
     -Dzgelsd=zgelsd_ zgelsd_ex.c <link line>
```

- On Windows* OS:

```
<cc> <compiler options>
     -Ddgesv=DGESV dgesv_ex.c <link line>
```

```
<cc> <compiler options>
     -Dzgelsd=ZGELSD zgelsd_ex.c <link line>
```

where *<cc>* is a C compiler, *dgesv_ex.c* and *zgelsd_ex.c* are oneMKL LAPACK C examples, and *<link line>* contains linker options and arguments, including the LAPACK library files required for your application.

To compile and build an example:

1. If needed, redefine the routine to avoid an unresolved external reference.
2. Run the `mklvars` script to set the required environment variables.
3. Build the application using an ANSI C compiler. See the following example lines for reference:
   - with Intel® C++ Compiler, you can use

     `icc -mkl cgesv_ex.c` on Linux* OS, or

     `icl /Qmkl cgesv_ex.c` on Windows* OS.

     See the *Linking Your Application with the Intel Math Kernel Library* section of the oneMKL Developer Guide for more information on linking with the oneMKL.
   - with third-party compilers, you can use

     `cc -fopenmp -m64 -I${MKLROOT}/include cgesv_ex.c -Wl,--no-as-needed -L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_core -lmkl_gnu_thread -lpthread -lm -ldl` to compile with the GNU* C/C++ compiler on Linux* OS, or

     `cl cgesv_ex.c /link mkl_intel_lp64_dll.lib mkl_core_dll.lib mkl_intel_thread_dll.lib libiomp5md.lib` to compile with the 64-bit version of the Microsoft* C/C++ compiler on Windows* OS.

NOTE The examples above are linking dynamically against 64-bit oneMKL library files. To create a custom link line, use the oneMKL Link Line Advisor or refer to the oneMKL Developer Guide, section *Linking Your Application with the Intel oneAPI Math Kernel Library*.

See Also

oneMKL Developer Guide Linux* OS section *Linking Your Application with the Intel oneAPI Math Kernel Library*
section *Linking Your Application with the Intel oneAPI Math Kernel Library*
oneMKL Developer Guide Windows* OS section *Linking Your Application with the Intel oneAPI Math Kernel Library*
section *Linking Your Application with the Intel oneAPI Math Kernel Library*
oneMKL Developer Guide Linux* OS section *Getting Started > Setting Environment Variables*
section *Getting Started > Setting Environment Variables*
oneMKL Developer Guide Windows* OS section *Getting Started > Setting Environment Variables*
section *Getting Started > Setting Environment Variables*
oneMKL Link Line Advisor

Getting Help and Support

oneMKL provides a product web site that offers timely and comprehensive product information, including product features, white papers, and technical articles. For the latest information, check: http://www.intel.com/software/products/support.

Intel also provides a support web site that contains a rich repository of self help information, including getting started tips, known product issues, product errata, license information, user forums, and more (visit http://www.intel.com/software/products/).

Every purchase of an Intel Software Development Product includes a year of support services, which provides priority customer support at our Online Service Center web site, http://www.intel.com/supporttickets.

To register your product, contact Intel, or seek product support, please visit http://www.intel.com/software/products/support.

LAPACK Examples

Routine	Description	Examples
?geev	Computes the eigenvalues and, optionally, the left and/or right eigenvectors of a general matrix.	cgeev dgeev sgeev zgeev
?gels	Uses *QR* or *LQ* factorization to solve an overdetermined or underdetermined linear system with a full rank matrix.	cgels dgels sgels zgels
?gelsd	Computes the minimum norm solution to a linear least squares problem using the singular value decomposition of *A* and a divide and conquer method.	cgelsd dgelsd sgelsd zgelsd
?gesdd	Computes the singular value decomposition of a general rectangular matrix using a divide and conquer algorithm.	cgesdd dgesdd sgesdd zgesdd
?gesv	Computes the solution to the system of linear equations with a square matrix *A* and multiple right-hand sides.	cgesv dgesv sgesv zgesv
?gesvd	Computes the singular value decomposition of a general rectangular matrix.	cgesvd dgesvd sgesvd zgesvd
?heev	Computes all the eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.	cheev zheev
?heevd	Computes all the eigenvalues and, optionally, all the eigenvectors of a complex Hermitian matrix using a divide and conquer algorithm.	cheevd zheevd
?heevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix using the Relatively Robust Representations.	cheevr zheevr

Routine	Description	Examples
?heevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.	cheevx zheevx
?hesv	Computes the solution to the system of linear equations with a Hermitian matrix *A* and multiple right-hand sides.	chesv zhesv
?posv	Computes the solution to the system of linear equations with a symmetric or Hermitian positive definite matrix *A* and multiple right-hand sides.	cposv dposv sposv zposv
?syev	Computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.	dsyev ssyev
?syevd	Computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric matrix using a divide and conquer algorithm.	dsyevd ssyevd
?syevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix using the Relatively Robust Representations.	dsyevr ssyevr
?syevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a symmetric matrix.	dsyevx ssyevx
?sysv	Computes the solution to the system of linear equations with a real or complex symmetric matrix *A* and multiple right-hand sides.	csysv dsysv ssysv zsysv

LAPACK Routines: Linear Equations

The Intel® oneAPI Math Kernel Library includes LAPACK routines for solving systems of linear equations, factoring and inverting matrices, and estimating condition numbers.

To solve a particular problem, you can call two or more **computational routines** or call a corresponding **driver routine** that combines several tasks in one call, such as `?gesv` for factoring and solving. For example, to solve a system of linear equations with a general matrix, call `?getrf` (*LU* factorization) and then `?getrs` (computing the solution). Call `?gerfs` to refine the solution and get the error bounds. Alternatively, use the driver routine `?gesvx`, which performs all these tasks in one call.

You can also find an appropriate routine using the **characteristics** of your data and the **operations** you need by using the oneMKL LAPACK Function Finding Advisor, which helps you find the routine that suits your needs best.

Caution
The standard LAPACK functions do not check the input data (matrices) for IEEE 754 floating point INFs or NANS. INFs and NANs will propagate through the computations and may cause unexpected results or instabilities. It is the user's responsibility to ensure that the input data do not contain INFs nor NaNs.

The LAPACKE (C interfaces to LAPACK) functions do check for NaNs in the input data (matrices) , and an error code is returned if a NAN is found.

Driver Routines for Solving Systems of Linear Equations

Routine	Description	Examples
?gesv	Computes the solution to the system of linear equations with a square matrix *A* and multiple right-hand sides.	cgesv dgesv sgesv zgesv
?posv	Computes the solution to the system of linear equations with a symmetric or Hermitian positive definite matrix *A* and multiple right-hand sides.	cposv dposv sposv zposv
?sysv	Computes the solution to the system of linear equations with a real or complex symmetric matrix *A* and multiple right-hand sides.	csysv dsysv ssysv zsysv
?hesv	Computes the solution to the system of linear equations with a Hermitian matrix *A* and multiple right-hand sides.	chesv zhesv

?gesv function

Computes the solution to the system of linear equations with a square coefficient matrix *A* and multiple right-hand sides.

NOTE
For a detailed description and reference information on this function, please visit:

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call cgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call zgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call dsgesv(n, nrhs, a, lda, ipiv, b, ldb, x, ldx, work, swork, iter, info)
call zcgesv(n, nrhs, a, lda, ipiv, b, ldb, x, ldx, work, swork, rwork, iter, info)
```

Calling from C:

```
sgesv(n, nrhs, a, lda, ipiv, b, ldb, info);
dgesv(n, nrhs, a, lda, ipiv, b, ldb, info);
cgesv(n, nrhs, a, lda, ipiv, b, ldb, info);
zgesv(n, nrhs, a, lda, ipiv, b, ldb, info);
dsgesv(n, nrhs, a, lda, ipiv, b, ldb, x, ldx, work, swork, iter, info)
zcgesv(n, nrhs, a, lda, ipiv, b, ldb, x, ldx, work, swork, rwork, iter, info)
```

C Interface

```
lapack_int LAPACKE_sgesv(int matrix_layout, lapack_int n, lapack_int nrhs, float* a,
lapack_int lda, lapack_int* ipiv, float* b, lapack_int ldb);
lapack_int  LAPACKE_dgesv(int matrix_layout, lapack_int n, lapack_int nrhs, double* a,
lapack_int lda, lapack_int* ipiv, double* b, lapack_int ldb);
lapack_int LAPACKE_cgesv(int matrix_layout, lapack_int n, lapack_int nrhs,
lapack_complex_float* a, lapack_int lda, lapack_int* ipiv, lapack_complex_float* b,
lapack_int ldb);
lapack_int LAPACKE_zgesv(int matrix_layout, lapack_int n, lapack_int nrhs,
lapack_complex_double* a, lapack_int lda, lapack_int* ipiv, lapack_complex_double* b,
lapack_int ldb);
lapack_int LAPACKE_dsgesv(int matrix_layout, lapack_int n, lapack_int nrhs, double * a,
lapack_int lda, lapack_int * ipiv, double * b, lapack_int ldb, double * x, lapack_int ldx,
lapack_int * iter);
lapack_int LAPACKE_zcgesv(int matrix_layout, lapack_int n, lapack_int nrhs,
lapack_complex_double * a, lapack_int lda, lapack_int * ipiv, lapack_complex_double * b,
lapack_int ldb, lapack_complex_double * x, lapack_int ldx, lapack_int * iter);
```

See Also

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CGESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The *LU* decomposition with partial pivoting and row interchanges is used to factor *A* as:

```
A = P*L*U
```

where

- *P* is a permutation matrix.
- *L* is a unit lower triangular.
- *U* is an upper triangular.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGESV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CGESV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a square matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*  (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
*  ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
*  ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
*  (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)
*
*  and B is the right-hand side matrix:
*
*  (  8.33, -7.32) ( -6.11, -3.81)
*  ( -6.18, -4.80) (  0.14, -7.71)
*  ( -5.71, -2.80) (  1.41,  3.40)
*  ( -1.60,  3.08) (  8.54, -4.05)
*
*  Description.
*  ============
*
*  The routine solves for X the system of linear equations A*X = B,
*  where A is an n-by-n matrix, the columns of matrix B are individual
*  right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as A = P*L*U, where P is a permutation matrix, L
*  is unit lower triangular, and U is upper triangular. The factored
*  form of A is then used to solve the system of equations A*X = B.
```

```
*
*  Example Program Results.
*  ========================
*
* CGESV Example Program Results
*
* Solution
* ( -1.09, -0.18) (  1.28,  1.21)
* (  0.97,  0.52) ( -0.22, -0.97)
* ( -0.20,  0.19) (  0.53,  1.36)
* ( -0.59,  0.92) (  2.22, -1.00)
*
* Details of LU factorization
* ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
* (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
* (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
* ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)
*
* Pivot indices
*      3      3      3      4
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX          A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $ ( 1.23,-5.50),(-2.14,-1.12),(-4.30,-7.10),( 1.27, 7.29),
     $ ( 7.91,-5.38),(-9.92,-0.79),(-6.47, 2.52),( 8.90, 6.92),
     $ (-9.80,-4.86),(-9.18,-1.12),(-6.51,-2.67),(-8.82, 1.25),
     $ (-7.32, 7.57),( 1.37, 0.43),(-5.86, 7.38),( 5.41, 5.37)
     $                 /
      DATA             B/
     $ ( 8.33,-7.32),(-6.18,-4.80),(-5.71,-2.80),(-1.60, 3.08),
     $ (-6.11,-3.81),( 0.14,-7.71),( 1.41, 3.40),( 8.54,-4.05)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CGESV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Executable Statements ..
      WRITE(*,*)'CGESV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL CGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*     Check for the exact singularity.
*
```

```
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element of the triangular factor of A,'
         WRITE(*,*)'U(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'A is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of LU factorization.
*
      CALL PRINT_MATRIX( 'Details of LU factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of CGESV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
```

```
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

CGESV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   CGESV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================
```

```
 CGESV Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGESV prototype */
extern void cgesv( int* n, int* nrhs, fcomplex* a, int* lda, int* ipiv,
                fcomplex* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        int ipiv[N];
        fcomplex a[LDA*N] = {
           { 1.23f, -5.50f}, {-2.14f, -1.12f}, {-4.30f, -7.10f}, { 1.27f,  7.29f},
           { 7.91f, -5.38f}, {-9.92f, -0.79f}, {-6.47f,  2.52f}, { 8.90f,  6.92f},
           {-9.80f, -4.86f}, {-9.18f, -1.12f}, {-6.51f, -2.67f}, {-8.82f,  1.25f},
           {-7.32f,  7.57f}, { 1.37f,  0.43f}, {-5.86f,  7.38f}, { 5.41f,  5.37f}
        };
        fcomplex b[LDB*NRHS] = {
           { 8.33f, -7.32f}, {-6.18f, -4.80f}, {-5.71f, -2.80f}, {-1.60f,  3.08f},
           {-6.11f, -3.81f}, { 0.14f, -7.71f}, { 1.41f,  3.40f}, { 8.54f, -4.05f}
        };
        /* Executable statements */
        printf( " CGESV Example Program Results\n" );
        /* Solve the equations A*X = B */
        cgesv( &n, &nrhs, a, &lda, ipiv, b, &ldb, &info );
```

```
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of CGESV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_cgesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesv Example.
   ======================
```

```
   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_cgesv (row-major, high-level) Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
```

```
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           { 1.23f, -5.50f}, { 7.91f, -5.38f}, {-9.80f, -4.86f}, {-7.32f,  7.57f},
           {-2.14f, -1.12f}, {-9.92f, -0.79f}, {-9.18f, -1.12f}, { 1.37f,  0.43f},
           {-4.30f, -7.10f}, {-6.47f,  2.52f}, {-6.51f, -2.67f}, {-5.86f,  7.38f},
           { 1.27f,  7.29f}, { 8.90f,  6.92f}, {-8.82f,  1.25f}, { 5.41f,  5.37f}
        };
        MKL_Complex8 b[LDB*N] = {
           { 8.33f, -7.32f}, {-6.11f, -3.81f},
           {-6.18f, -4.80f}, { 0.14f, -7.71f},
           {-5.71f, -2.80f}, { 1.41f,  3.40f},
           {-1.60f,  3.08f}, { 8.54f, -4.05f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgesv( LAPACK_ROW_MAJOR, n, nrhs, a, lda, ipiv, b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_cgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
```

```
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_cgesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================
```

```
 LAPACKE_cgesv (column-major, high-level) Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           { 1.23f, -5.50f}, {-2.14f, -1.12f}, {-4.30f, -7.10f}, { 1.27f,  7.29f},
           { 7.91f, -5.38f}, {-9.92f, -0.79f}, {-6.47f,  2.52f}, { 8.90f,  6.92f},
           {-9.80f, -4.86f}, {-9.18f, -1.12f}, {-6.51f, -2.67f}, {-8.82f,  1.25f},
           {-7.32f,  7.57f}, { 1.37f,  0.43f}, {-5.86f,  7.38f}, { 5.41f,  5.37f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           { 8.33f, -7.32f}, {-6.18f, -4.80f}, {-5.71f, -2.80f}, {-1.60f,  3.08f},
           {-6.11f, -3.81f}, { 0.14f, -7.71f}, { 1.41f,  3.40f}, { 8.54f, -4.05f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgesv( LAPACK_COL_MAJOR, n, nrhs, a, lda, ipiv, b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
```

```
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_cgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

DGESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The *LU* decomposition with partial pivoting and row interchanges is used to factor *A* as:

```
A = P*L*U
```

where

- *P* is a permutation matrix.
- *L* is a unit lower triangular.
- *U* is an upper triangular.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGESV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DGESV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a square matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*    6.80  -6.05  -0.45   8.32  -9.67
*   -2.11  -3.30   2.58   2.71  -5.14
*    5.66   5.36  -2.70   4.35  -7.26
*    5.97  -4.44   0.27  -7.17   6.08
*    8.23   1.08   9.04   2.14  -6.87
*
*  and B is the right-hand side matrix:
*
*    4.02  -1.56   9.81
*    6.19   4.00  -4.09
*   -8.22  -8.67  -4.57
*   -7.57   1.75  -8.61
*   -3.03   2.86   8.99
*
*  Description.
*  ============
*
*  The routine solves for X the system of linear equations A*X = B,
*  where A is an n-by-n matrix, the columns of matrix B are individual
```

```
*  right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as A = P*L*U, where P is a permutation matrix, L
*  is unit lower triangular, and U is upper triangular. The factored
*  form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* DGESV Example Program Results
*
* Solution
*  -0.80  -0.39   0.96
*  -0.70  -0.55   0.22
*   0.59   0.84   1.90
*   1.32  -0.10   5.36
*   0.57   0.11   4.04
*
* Details of LU factorization
*   8.23   1.08   9.04   2.14  -6.87
*   0.83  -6.94  -7.92   6.55  -3.99
*   0.69  -0.67 -14.18   7.24  -5.19
*   0.73   0.75   0.02 -13.82  14.19
*  -0.26   0.44  -0.59  -0.34  -3.43
*
* Pivot indices
*      5      5      3      4      5
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $  6.80,-2.11, 5.66, 5.97, 8.23,
     $ -6.05,-3.30, 5.36,-4.44, 1.08,
     $ -0.45, 2.58,-2.70, 0.27, 9.04,
     $  8.32, 2.71, 4.35,-7.17, 2.14,
     $ -9.67,-5.14,-7.26, 6.08,-6.87
     $                 /
      DATA             B/
     $  4.02, 6.19,-8.22,-7.57,-3.03,
     $ -1.56, 4.00,-8.67, 1.75, 2.86,
     $  9.81,-4.09,-4.57,-8.61, 8.99
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         DGESV
```

```
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Executable Statements ..
      WRITE(*,*)'DGESV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element of the triangular factor of A,'
         WRITE(*,*)'U(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'A is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of LU factorization.
*
      CALL PRINT_MATRIX( 'Details of LU factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of DGESV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
```

```
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

DGESV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   DGESV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99

   Description.
   ============
```

```
  The routine solves for X the system of linear equations A*X = B,
  where A is an n-by-n matrix, the columns of matrix B are individual
  right-hand sides, and the columns of X are the corresponding
  solutions.

  The LU decomposition with partial pivoting and row interchanges is
  used to factor A as A = P*L*U, where P is a permutation matrix, L
  is unit lower triangular, and U is upper triangular. The factored
  form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 DGESV Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04

 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>

/* DGESV prototype */
extern void dgesv( int* n, int* nrhs, double* a, int* lda, int* ipiv,
                double* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        int ipiv[N];
        double a[LDA*N] = {
            6.80, -2.11,  5.66,  5.97,  8.23,
           -6.05, -3.30,  5.36, -4.44,  1.08,
           -0.45,  2.58, -2.70,  0.27,  9.04,
```

```
            8.32,  2.71,  4.35, -7.17,  2.14,
           -9.67, -5.14, -7.26,  6.08, -6.87
        };
        double b[LDB*NRHS] = {
            4.02,  6.19, -8.22, -7.57, -3.03,
           -1.56,  4.00, -8.67,  1.75,  2.86,
            9.81, -4.09, -4.57, -8.61,  8.99
        };
        /* Executable statements */
        printf( " DGESV Example Program Results\n" );
        /* Solve the equations A*X = B */
        dgesv( &n, &nrhs, a, &lda, ipiv, b, &ldb, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of DGESV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACK_dgesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_dgesv (row-major, high-level) Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04

 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
```

```
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        double a[LDA*N] = {
            6.80, -6.05, -0.45,  8.32, -9.67,
           -2.11, -3.30,  2.58,  2.71, -5.14,
            5.66, 5.36, -2.70,  4.35, -7.26,
            5.97, -4.44,  0.27, -7.17, 6.08,
            8.23, 1.08,  9.04,  2.14, -6.87
        };
        double b[LDB*N] = {
            4.02, -1.56, 9.81,
            6.19,  4.00, -4.09,
           -8.22, -8.67, -4.57,
           -7.57,  1.75, -8.61,
           -3.03,  2.86, 8.99
        };
        /* Executable statements */
        printf( "LAPACKE_dgesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgesv( LAPACK_ROW_MAJOR, n, nrhs, a, lda, ipiv,
                              b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
```

```
        exit( 0 );
} /* End of LAPACKE_dgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_dgesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
```

```
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_dgesv (column-major, high-level) Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04

 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
```

```
        MKL_INT ipiv[N];
        double a[LDA*N] = {
            6.80, -2.11,  5.66,  5.97,  8.23,
           -6.05, -3.30,  5.36, -4.44,  1.08,
           -0.45,  2.58, -2.70,  0.27,  9.04,
            8.32,  2.71,  4.35, -7.17,  2.14,
           -9.67, -5.14, -7.26,  6.08, -6.87
        };
        double b[LDB*NRHS] = {
            4.02,  6.19, -8.22, -7.57, -3.03,
           -1.56,  4.00, -8.67,  1.75,  2.86,
            9.81, -4.09, -4.57, -8.61,  8.99
        };
        /* Executable statements */
        printf( "LAPACKE_dgesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgesv( LAPACK_COL_MAJOR, n, nrhs, a, lda, ipiv,
                              b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_dgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

SGESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The *LU* decomposition with partial pivoting and row interchanges is used to factor *A* as:

```
A = P*L*U
```

where

- *P* is a permutation matrix.
- *L* is a unit lower triangular.
- *U* is an upper triangular.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGESV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
```

```
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  ============================================================================
*
*  SGESV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a square matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*    6.80  -6.05  -0.45   8.32  -9.67
*   -2.11  -3.30   2.58   2.71  -5.14
*    5.66   5.36  -2.70   4.35  -7.26
*    5.97  -4.44   0.27  -7.17   6.08
*    8.23   1.08   9.04   2.14  -6.87
*
*  and B is the right-hand side matrix:
*
*    4.02  -1.56   9.81
*    6.19   4.00  -4.09
*   -8.22  -8.67  -4.57
*   -7.57   1.75  -8.61
*   -3.03   2.86   8.99
*
*  Description.
*  ============
*
*  The routine solves for X the system of linear equations A*X = B,
*  where A is an n-by-n matrix, the columns of matrix B are individual
*  right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as A = P*L*U, where P is a permutation matrix, L
*  is unit lower triangular, and U is upper triangular. The factored
*  form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* SGESV Example Program Results
*
* Solution
*  -0.80  -0.39   0.96
*  -0.70  -0.55   0.22
*   0.59   0.84   1.90
*   1.32  -0.10   5.36
*   0.57   0.11   4.04
*
* Details of LU factorization
*   8.23   1.08   9.04   2.14  -6.87
*   0.83  -6.94  -7.92   6.55  -3.99
*   0.69  -0.67 -14.18   7.24  -5.19
*   0.73   0.75   0.02 -13.82  14.19
*  -0.26   0.44  -0.59  -0.34  -3.43
*
* Pivot indices
```

```
*      5      5      3      4      5
*  ================================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      REAL             A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $  6.80,-2.11, 5.66, 5.97, 8.23,
     $ -6.05,-3.30, 5.36,-4.44, 1.08,
     $ -0.45, 2.58,-2.70, 0.27, 9.04,
     $  8.32, 2.71, 4.35,-7.17, 2.14,
     $ -9.67,-5.14,-7.26, 6.08,-6.87
     $                 /
      DATA             B/
     $  4.02, 6.19,-8.22,-7.57,-3.03,
     $ -1.56, 4.00,-8.67, 1.75, 2.86,
     $  9.81,-4.09,-4.57,-8.61, 8.99
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         SGESV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Executable Statements ..
      WRITE(*,*)'SGESV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL SGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element of the triangular factor of A,'
         WRITE(*,*)'U(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'A is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of LU factorization.
*
      CALL PRINT_MATRIX( 'Details of LU factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
```

```
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of SGESV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

SGESV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
```

```
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SGESV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 SGESV Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04

 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43
```

```
 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>

/* SGESV prototype */
extern void sgesv( int* n, int* nrhs, float* a, int* lda, int* ipiv,
                float* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        int ipiv[N];
        float a[LDA*N] = {
            6.80f, -2.11f,  5.66f,  5.97f,  8.23f,
           -6.05f, -3.30f,  5.36f, -4.44f,  1.08f,
           -0.45f,  2.58f, -2.70f,  0.27f,  9.04f,
            8.32f,  2.71f,  4.35f, -7.17f,  2.14f,
           -9.67f, -5.14f, -7.26f,  6.08f, -6.87f
        };
        float b[LDB*NRHS] = {
            4.02f,  6.19f, -8.22f, -7.57f, -3.03f,
           -1.56f,  4.00f, -8.67f,  1.75f,  2.86f,
            9.81f, -4.09f, -4.57f, -8.61f,  8.99f
        };
        /* Executable statements */
        printf( " SGESV Example Program Results\n" );
        /* Solve the equations A*X = B */
        sgesv( &n, &nrhs, a, &lda, ipiv, b, &ldb, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of SGESV Example */
```

```
/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACK_sgesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99
```

```
  Description.
  ============

  The routine solves for X the system of linear equations A*X = B,
  where A is an n-by-n matrix, the columns of matrix B are individual
  right-hand sides, and the columns of X are the corresponding
  solutions.

  The LU decomposition with partial pivoting and row interchanges is
  used to factor A as A = P*L*U, where P is a permutation matrix, L
  is unit lower triangular, and U is upper triangular. The factored
  form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_sgesv (row-major, high-level) Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04

 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        float a[LDA*N] = {
            6.80f, -6.05f, -0.45f,  8.32f, -9.67f,
```

```
           -2.11f, -3.30f,  2.58f,  2.71f, -5.14f,
            5.66f, 5.36f, -2.70f,  4.35f, -7.26f,
            5.97f, -4.44f,  0.27f, -7.17f, 6.08f,
            8.23f, 1.08f,  9.04f,  2.14f, -6.87f
        };
        float b[LDB*N] = {
            4.02f, -1.56f, 9.81f,
            6.19f,  4.00f, -4.09f,
           -8.22f, -8.67f, -4.57f,
           -7.57f,  1.75f, -8.61f,
           -3.03f,  2.86f, 8.99f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgesv( LAPACK_ROW_MAJOR, n, nrhs, a, lda, ipiv,
                              b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_sgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACK_sgesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
```

```
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     6.80  -6.05  -0.45   8.32  -9.67
    -2.11  -3.30   2.58   2.71  -5.14
     5.66   5.36  -2.70   4.35  -7.26
     5.97  -4.44   0.27  -7.17   6.08
     8.23   1.08   9.04   2.14  -6.87

   and B is the right-hand side matrix:

     4.02  -1.56   9.81
     6.19   4.00  -4.09
    -8.22  -8.67  -4.57
    -7.57   1.75  -8.61
    -3.03   2.86   8.99

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_sgesv (column-major, high-level) Example Program Results

 Solution
  -0.80  -0.39   0.96
  -0.70  -0.55   0.22
   0.59   0.84   1.90
   1.32  -0.10   5.36
   0.57   0.11   4.04
```

```
 Details of LU factorization
   8.23   1.08   9.04   2.14  -6.87
   0.83  -6.94  -7.92   6.55  -3.99
   0.69  -0.67 -14.18   7.24  -5.19
   0.73   0.75   0.02 -13.82  14.19
  -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        float a[LDA*N] = {
            6.80f, -2.11f,  5.66f,  5.97f,  8.23f,
           -6.05f, -3.30f,  5.36f, -4.44f,  1.08f,
           -0.45f,  2.58f, -2.70f,  0.27f,  9.04f,
            8.32f,  2.71f,  4.35f, -7.17f,  2.14f,
           -9.67f, -5.14f, -7.26f,  6.08f, -6.87f
        };
        float b[LDB*NRHS] = {
            4.02f,  6.19f, -8.22f, -7.57f, -3.03f,
           -1.56f,  4.00f, -8.67f,  1.75f,  2.86f,
            9.81f, -4.09f, -4.57f, -8.61f,  8.99f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgesv( LAPACK_COL_MAJOR, n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
```

```
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_sgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

ZGESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The *LU* decomposition with partial pivoting and row interchanges is used to factor *A* as:

```
A = P*L*U
```

where

- *P* is a permutation matrix.
- *L* is a unit lower triangular.
- *U* is an upper triangular.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major[1] Data Layout
- Example program in C for Column Major[1] Data Layout

See Also

?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGESV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZGESV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a square matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*  (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
*  ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
*  ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
*  (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)
*
*  and B is the right-hand side matrix:
*
*  (  8.33, -7.32) ( -6.11, -3.81)
*  ( -6.18, -4.80) (  0.14, -7.71)
*  ( -5.71, -2.80) (  1.41,  3.40)
*  ( -1.60,  3.08) (  8.54, -4.05)
*
*  Description.
*  ============
*
*  The routine solves for X the system of linear equations A*X = B,
*  where A is an n-by-n matrix, the columns of matrix B are individual
*  right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as A = P*L*U, where P is a permutation matrix, L
*  is unit lower triangular, and U is upper triangular. The factored
*  form of A is then used to solve the system of equations A*X = B.
```

```
*
*  Example Program Results.
*  ========================
*
* ZGESV Example Program Results
*
* Solution
* ( -1.09, -0.18) (  1.28,  1.21)
* (  0.97,  0.52) ( -0.22, -0.97)
* ( -0.20,  0.19) (  0.53,  1.36)
* ( -0.59,  0.92) (  2.22, -1.00)
*
* Details of LU factorization
* ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
* (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
* (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
* ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)
*
* Pivot indices
*      3      3      3      4
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $ ( 1.23,-5.50),(-2.14,-1.12),(-4.30,-7.10),( 1.27, 7.29),
     $ ( 7.91,-5.38),(-9.92,-0.79),(-6.47, 2.52),( 8.90, 6.92),
     $ (-9.80,-4.86),(-9.18,-1.12),(-6.51,-2.67),(-8.82, 1.25),
     $ (-7.32, 7.57),( 1.37, 0.43),(-5.86, 7.38),( 5.41, 5.37)
     $                 /
      DATA             B/
     $ ( 8.33,-7.32),(-6.18,-4.80),(-5.71,-2.80),(-1.60, 3.08),
     $ (-6.11,-3.81),( 0.14,-7.71),( 1.41, 3.40),( 8.54,-4.05)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZGESV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Executable Statements ..
      WRITE(*,*)'ZGESV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL ZGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*     Check for the exact singularity.
*
```

```
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element of the triangular factor of A,'
         WRITE(*,*)'U(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'A is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of LU factorization.
*
      CALL PRINT_MATRIX( 'Details of LU factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of ZGESV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
```

```
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

ZGESV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   ZGESV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================
```

```
 ZGESV Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGESV prototype */
extern void zgesv( int* n, int* nrhs, dcomplex* a, int* lda, int* ipiv,
                dcomplex* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        int ipiv[N];
        dcomplex a[LDA*N] = {
           { 1.23, -5.50}, {-2.14, -1.12}, {-4.30, -7.10}, { 1.27,  7.29},
           { 7.91, -5.38}, {-9.92, -0.79}, {-6.47,  2.52}, { 8.90,  6.92},
           {-9.80, -4.86}, {-9.18, -1.12}, {-6.51, -2.67}, {-8.82,  1.25},
           {-7.32,  7.57}, { 1.37,  0.43}, {-5.86,  7.38}, { 5.41,  5.37}
        };
        dcomplex b[LDB*NRHS] = {
           { 8.33, -7.32}, {-6.18, -4.80}, {-5.71, -2.80}, {-1.60,  3.08},
           {-6.11, -3.81}, { 0.14, -7.71}, { 1.41,  3.40}, { 8.54, -4.05}
        };
        /* Executable statements */
        printf( " ZGESV Example Program Results\n" );
        /* Solve the equations A*X = B */
        zgesv( &n, &nrhs, a, &lda, ipiv, b, &ldb, &info );
```

```
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of ZGESV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zgesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesv Example.
   ======================
```

```
   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_zgesv (row-major, high-level) Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
```

```
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           { 1.23, -5.50}, { 7.91, -5.38}, {-9.80, -4.86}, {-7.32,  7.57},
           {-2.14, -1.12}, {-9.92, -0.79}, {-9.18, -1.12}, { 1.37,  0.43},
           {-4.30, -7.10}, {-6.47,  2.52}, {-6.51, -2.67}, {-5.86,  7.38},
           { 1.27,  7.29}, { 8.90,  6.92}, {-8.82,  1.25}, { 5.41,  5.37}
        };
        MKL_Complex16 b[LDB*N] = {
           { 8.33, -7.32}, {-6.11, -3.81},
           {-6.18, -4.80}, { 0.14, -7.71},
           {-5.71, -2.80}, { 1.41,  3.40},
           {-1.60,  3.08}, { 8.54, -4.05}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgesv( LAPACK_ROW_MAJOR, n, nrhs, a, lda, ipiv, b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
```

```
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zgesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a square matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  1.23, -5.50) (  7.91, -5.38) ( -9.80, -4.86) ( -7.32,  7.57)
   ( -2.14, -1.12) ( -9.92, -0.79) ( -9.18, -1.12) (  1.37,  0.43)
   ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
   (  1.27,  7.29) (  8.90,  6.92) ( -8.82,  1.25) (  5.41,  5.37)

   and B is the right-hand side matrix:

   (  8.33, -7.32) ( -6.11, -3.81)
   ( -6.18, -4.80) (  0.14, -7.71)
   ( -5.71, -2.80) (  1.41,  3.40)
   ( -1.60,  3.08) (  8.54, -4.05)

   Description.
   ============

   The routine solves for X the system of linear equations A*X = B,
   where A is an n-by-n matrix, the columns of matrix B are individual
   right-hand sides, and the columns of X are the corresponding
   solutions.

   The LU decomposition with partial pivoting and row interchanges is
   used to factor A as A = P*L*U, where P is a permutation matrix, L
   is unit lower triangular, and U is upper triangular. The factored
   form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================
```

```
 LAPACKE_zgesv (column-major, high-level) Example Program Results

 Solution
 ( -1.09, -0.18) (  1.28,  1.21)
 (  0.97,  0.52) ( -0.22, -0.97)
 ( -0.20,  0.19) (  0.53,  1.36)
 ( -0.59,  0.92) (  2.22, -1.00)

 Details of LU factorization
 ( -4.30, -7.10) ( -6.47,  2.52) ( -6.51, -2.67) ( -5.86,  7.38)
 (  0.49,  0.47) ( 12.26, -3.57) ( -7.87, -0.49) ( -0.98,  6.71)
 (  0.25, -0.15) ( -0.60, -0.37) (-11.70, -4.64) ( -1.35,  1.38)
 ( -0.83, -0.32) (  0.05,  0.58) (  0.93, -0.50) (  2.66,  7.86)

 Pivot indices
      3      3      3      4
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           { 1.23, -5.50}, {-2.14, -1.12}, {-4.30, -7.10}, { 1.27,  7.29},
           { 7.91, -5.38}, {-9.92, -0.79}, {-6.47,  2.52}, { 8.90,  6.92},
           {-9.80, -4.86}, {-9.18, -1.12}, {-6.51, -2.67}, {-8.82,  1.25},
           {-7.32,  7.57}, { 1.37,  0.43}, {-5.86,  7.38}, { 5.41,  5.37}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           { 8.33, -7.32}, {-6.18, -4.80}, {-5.71, -2.80}, {-1.60,  3.08},
           {-6.11, -3.81}, { 0.14, -7.71}, { 1.41,  3.40}, { 8.54, -4.05}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgesv( LAPACK_COL_MAJOR, n, nrhs, a, lda, ipiv, b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The diagonal element of the triangular factor of A,\n" );
                printf( "U(%i,%i) is zero, so that A is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
```

```
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of LU factorization */
        print_matrix( "Details of LU factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zgesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

?posv function

Computes the solution to the system of linear equations with a symmetric or Hermitian positive-definite coefficient matrix *A* and multiple right-hand sides.

NOTE
For a detailed description and reference information on this function, please visit:

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sposv(uplo, n, nrhs, a, lda, b, ldb, info)
call dposv(uplo, n, nrhs, a, lda, b, ldb, info)
call cposv(uplo, n, nrhs, a, lda, b, ldb, info)
call zposv(uplo, n, nrhs, a, lda, b, ldb, info)
call dsposv(uplo, n, nrhs, a, lda, b, ldb, x, ldx, work, swork, iter, info)
call zcposv(uplo, n, nrhs, a, lda, b, ldb, x, ldx, work, swork, rwork, iter, info)
```

Calling from C:

```
sposv(uplo, n, nrhs, a, lda, b, ldb, info);
dposv(uplo, n, nrhs, a, lda, b, ldb, info);
cposv(uplo, n, nrhs, a, lda, b, ldb, info);
zposv(uplo, n, nrhs, a, lda, b, ldb, info);
dsposv(uplo, n, nrhs, a, lda, b, ldb, x, ldx, work, swork, iter, info)
zcposv(uplo, n, nrhs, a, lda, b, ldb, x, ldx, work, swork, rwork, iter, info)
```

C Interface

```
lapack_int LAPACKE_sposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
float* a, lapack_int lda, float* b, lapack_int ldb);
lapack_int LAPACKE_dposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
double* a, lapack_int lda, double* b, lapack_int ldb);
lapack_int LAPACKE_cposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_float* a, lapack_int lda, lapack_complex_float* b, lapack_int ldb);
lapack_int LAPACKE_zposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_double* a, lapack_int lda, lapack_complex_double* b, lapack_int ldb);
lapack_int LAPACKE_dsposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
double * a, lapack_int lda, double * b, lapack_int ldb, double * x, lapack_int ldx,
lapack_int * iter);
lapack_int LAPACKE_zcposv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_double * a, lapack_int lda, lapack_complex_double * b, lapack_int ldb,
lapack_complex_double * x, lapack_int ldx, lapack_int * iter);
```

See Also

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CPOSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square Hermitian positive-definite matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The Cholesky decomposition is used to factor *A* as:

$A = U^H * U$, if `uplo = 'U'` or $A = L * L^H$, if `uplo = 'L'`

where

- *U* is an upper triangular matrix.
- *L* is a lower triangular matrix.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CPOSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CPOSV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a Hermitian positive-definite matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*  (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
*  (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
*  ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
*  ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)
*
*  and B is the right-hand side matrix:
*
*  ( -2.94,  5.79) (  8.44,  3.07)
*  (  8.12, -9.12) (  1.00, -4.62)
*  (  9.09, -5.03) (  3.64, -2.33)
*  (  7.36,  6.77) (  8.04,  2.87)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations
*  A*X = B, where A is an n-by-n Hermitian positive-definite
*  matrix, the columns of matrix B are individual right-hand sides,
*  and the columns of X are the corresponding solutions.
*
*  The Cholesky decomposition is used to factor A as
*  A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
*  where U is an upper triangular matrix and L is a lower triangular matrix.
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
```

```
* CPOSV Example Program Results
*
* Solution
* (  0.80,  1.62) (  2.52,  0.61)
* (  1.26, -1.78) (  0.01, -1.38)
* (  3.38, -0.29) (  2.42, -0.52)
* (  3.46,  2.92) (  3.77,  1.37)
*
* Details of Cholesky factorization
* (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
* (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
* ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
* ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      COMPLEX          A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $ ( 5.96, 0.00),( 0.40, 1.19),(-0.83, 0.48),(-0.57,-0.40),
     $ ( 0.00, 0.00),( 7.95, 0.00),( 0.33,-0.09),( 0.22,-0.74),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 4.43, 0.00),(-1.09,-0.32),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 3.46, 0.00)
     $                 /
      DATA             B/
     $ (-2.94, 5.79),( 8.12,-9.12),( 9.09,-5.03),( 7.36, 6.77),
     $ ( 8.44, 3.07),( 1.00,-4.62),( 3.64,-2.33),( 8.04, 2.87)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CPOSV
      EXTERNAL         PRINT_MATRIX
*
*     .. Executable Statements ..
      WRITE(*,*)'CPOSV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL CPOSV( 'Lower', N, NRHS, A, LDA, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The leading minor of order ',INFO,' is not positive'
         WRITE(*,*)'definite; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
```

```
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of Cholesky factorization.
*
      CALL PRINT_MATRIX( 'Details of Cholesky factorization', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of CPOSV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

CPOSV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CPOSV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:
```

```
   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 CPOSV Example Program Results

 Solution
 (  0.80,  1.62) (  2.52,  0.61)
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CPOSV prototype */
extern void cposv( char* uplo, int* n, int* nrhs, fcomplex* a, int* lda,
                fcomplex* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );

/* Parameters */
#define N 4
```

```
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        fcomplex a[LDA*N] = {
           { 5.96f,  0.00f}, { 0.40f,  1.19f}, {-0.83f,  0.48f}, {-0.57f, -0.40f},
           { 0.00f,  0.00f}, { 7.95f,  0.00f}, { 0.33f, -0.09f}, { 0.22f, -0.74f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 4.43f,  0.00f}, {-1.09f, -0.32f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 3.46f,  0.00f}
        };
        fcomplex b[LDB*NRHS] = {
           {-2.94f,  5.79f}, { 8.12f, -9.12f}, { 9.09f, -5.03f}, { 7.36f,  6.77f},
           { 8.44f,  3.07f}, { 1.00f, -4.62f}, { 3.64f, -2.33f}, { 8.04f,  2.87f}
        };
        /* Executable statements */
        printf( " CPOSV Example Program Results\n" );
        /* Solve the equations A*X = B */
        cposv( "Lower", &n, &nrhs, a, &lda, b, &ldb, &info );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of CPOSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_cposv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
```

```
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_cposv (row-major, high-level) Example Program Results

 Solution
 (  0.80,  1.62) (  2.52,  0.61)
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
```

```
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex8 a[LDA*N] = {
           { 5.96f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           { 0.40f,  1.19f}, { 7.95f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-0.83f,  0.48f}, { 0.33f, -0.09f}, { 4.43f,  0.00f}, { 0.00f,  0.00f},
           {-0.57f, -0.40f}, { 0.22f, -0.74f}, {-1.09f, -0.32f}, { 3.46f,  0.00f}
        };
        MKL_Complex8 b[LDB*N] = {
           {-2.94f,  5.79f}, { 8.44f,  3.07f},
           { 8.12f, -9.12f}, { 1.00f, -4.62f},
           { 9.09f, -5.03f}, { 3.64f, -2.33f},
           { 7.36f,  6.77f}, { 8.04f,  2.87f}
        };
        /* Executable statements */
        printf( "LAPACKE_cposv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cposv( LAPACK_ROW_MAJOR, 'L', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

LAPACK_cposv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

LAPACKE_cposv (column-major, high-level) Example Program Results

Solution
(  0.80,  1.62) (  2.52,  0.61)
```

```
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex8 a[LDA*N] = {
           { 5.96f,  0.00f}, { 0.40f,  1.19f}, {-0.83f,  0.48f}, {-0.57f, -0.40f},
           { 0.00f,  0.00f}, { 7.95f,  0.00f}, { 0.33f, -0.09f}, { 0.22f, -0.74f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 4.43f,  0.00f}, {-1.09f, -0.32f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 3.46f,  0.00f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           {-2.94f,  5.79f}, { 8.12f, -9.12f}, { 9.09f, -5.03f}, { 7.36f,  6.77f},
           { 8.44f,  3.07f}, { 1.00f, -4.62f}, { 3.64f, -2.33f}, { 8.04f,  2.87f}
        };
        /* Executable statements */
        printf( "LAPACKE_cposv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cposv( LAPACK_COL_MAJOR, 'L', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

DPOSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the real system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric positive-definite matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The Cholesky decomposition is used to factor *A* as:

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

where

- *U* is an upper triangular matrix.
- *L* is a lower triangular matrix.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DPOSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
```

```
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DPOSV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a symmetric positive-definite matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*    3.14   0.17  -0.90   1.65  -0.72
*    0.17   0.79   0.83  -0.65   0.28
*   -0.90   0.83   4.53  -3.70   1.60
*    1.65  -0.65  -3.70   5.32  -1.37
*   -0.72   0.28   1.60  -1.37   1.98
*
*  and B is the right-hand side matrix:
*
*   -7.29   6.11   0.59
*    9.25   2.90   8.88
*    5.99  -5.05   7.57
*   -1.94  -3.80   5.57
*   -8.30   9.66  -1.67
*
*  Description.
*  ============
*
*  The routine solves for X the real system of linear equations
*  A*X = B, where A is an n-by-n symmetric positive-definite
*  matrix, the columns of matrix B are individual right-hand sides,
*  and the columns of X are the corresponding solutions.
*
*  The Cholesky decomposition is used to factor A as
*  A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
*  where U is an upper triangular matrix and L is a lower triangular matrix.
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* DPOSV Example Program Results
*
* Solution
*  -6.02   3.95  -3.14
*  15.62   4.32  13.05
*   3.02  -8.25   4.91
*   3.25  -4.83   6.11
*  -8.78   9.04  -3.57
*
```

```
* Details of Cholesky factorization
*   1.77   0.10  -0.51   0.93  -0.41
*   0.00   0.88   0.99  -0.84   0.36
*   0.00   0.00   1.81  -1.32   0.57
*   0.00   0.00   0.00   1.42   0.05
*   0.00   0.00   0.00   0.00   1.16
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $  3.14, 0.00, 0.00, 0.00, 0.00,
     $  0.17, 0.79, 0.00, 0.00, 0.00,
     $ -0.90, 0.83, 4.53, 0.00, 0.00,
     $  1.65,-0.65,-3.70, 5.32, 0.00,
     $ -0.72, 0.28, 1.60,-1.37, 1.98
     $                  /
      DATA             B/
     $ -7.29, 9.25, 5.99,-1.94,-8.30,
     $  6.11, 2.90,-5.05,-3.80, 9.66,
     $  0.59, 8.88, 7.57, 5.57,-1.67
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DPOSV
      EXTERNAL         PRINT_MATRIX
*
*     .. Executable Statements ..
      WRITE(*,*)'DPOSV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL DPOSV( 'Upper', N, NRHS, A, LDA, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The leading minor of order ',INFO,' is not positive'
         WRITE(*,*)'definite; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of Cholesky factorization.
*
      CALL PRINT_MATRIX( 'Details of Cholesky factorization', N, N, A,
```

```
     $                   LDA )
      STOP
      END
*
*     End of DPOSV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DPOSV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DPOSV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
```

```
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 DPOSV Example Program Results

 Solution
  -6.02   3.95  -3.14
  15.62   4.32  13.05
   3.02  -8.25   4.91
   3.25  -4.83   6.11
  -8.78   9.04  -3.57

 Details of Cholesky factorization
   1.77   0.10  -0.51   0.93  -0.41
   0.00   0.88   0.99  -0.84   0.36
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
#include <stdlib.h>
#include <stdio.h>

/* DPOSV prototype */
extern void dposv( char* uplo, int* n, int* nrhs, double* a, int* lda,
                double* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
```

```
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        double a[LDA*N] = {
            3.14,  0.00,  0.00,  0.00,  0.00,
            0.17,  0.79,  0.00,  0.00,  0.00,
           -0.90,  0.83,  4.53,  0.00,  0.00,
            1.65, -0.65, -3.70,  5.32,  0.00,
           -0.72,  0.28,  1.60, -1.37,  1.98
        };
        double b[LDB*NRHS] = {
           -7.29,  9.25,  5.99, -1.94, -8.30,
            6.11,  2.90, -5.05, -3.80,  9.66,
            0.59,  8.88,  7.57,  5.57, -1.67
        };
        /* Executable statements */
        printf( " DPOSV Example Program Results\n" );
        /* Solve the equations A*X = B */
        dposv( "Upper", &n, &nrhs, a, &lda, b, &ldb, &info );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of DPOSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dposv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
```

```
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_dposv (row-major, high-level) Example Program Results

 Solution
  -6.02   3.95  -3.14
  15.62   4.32  13.05
   3.02  -8.25   4.91
   3.25  -4.83   6.11
  -8.78   9.04  -3.57

 Details of Cholesky factorization
   1.77   0.10  -0.51   0.93  -0.41
   0.00   0.88   0.99  -0.84   0.36
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
```

```
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        double a[LDA*N] = {
            3.14,  0.17, -0.90, 1.65, -0.72,
            0.00,  0.79,  0.83, -0.65,  0.28,
            0.00,  0.00,  4.53, -3.70,  1.60,
            0.00,  0.00,  0.00, 5.32, -1.37,
            0.00,  0.00,  0.00, 0.00,  1.98
        };
        double b[LDB*N] = {
           -7.29,  6.11,  0.59,
            9.25,  2.90,  8.88,
            5.99, -5.05,  7.57,
           -1.94, -3.80,  5.57,
           -8.30,  9.66, -1.67
        };
        /* Executable statements */
        printf( "LAPACKE_dposv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dposv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACK_dposv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_dposv (column-major, high-level) Example Program Results
```

```
 Solution
  -6.02   3.95  -3.14
  15.62   4.32  13.05
   3.02  -8.25   4.91
   3.25  -4.83   6.11
  -8.78   9.04  -3.57

 Details of Cholesky factorization
   1.77   0.10  -0.51   0.93  -0.41
   0.00   0.88   0.99  -0.84   0.36
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        double a[LDA*N] = {
            3.14,  0.00,  0.00,  0.00,  0.00,
            0.17,  0.79,  0.00,  0.00,  0.00,
           -0.90,  0.83,  4.53,  0.00,  0.00,
            1.65, -0.65, -3.70,  5.32,  0.00,
           -0.72,  0.28,  1.60, -1.37,  1.98
        };
        double b[LDB*NRHS] = {
           -7.29,  9.25,  5.99, -1.94, -8.30,
            6.11,  2.90, -5.05, -3.80,  9.66,
            0.59,  8.88,  7.57,  5.57, -1.67
        };
        /* Executable statements */
        printf( "LAPACKE_dposv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dposv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
```

```
        exit( 0 );
} /* End of LAPACKE_dposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SPOSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the real system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric positive-definite matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The Cholesky decomposition is used to factor *A* as:

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

where

- *U* is an upper triangular matrix.
- *L* is a lower triangular matrix.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SPOSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SPOSV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a symmetric positive-definite matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*    3.14   0.17  -0.90   1.65  -0.72
*    0.17   0.79   0.83  -0.65   0.28
*   -0.90   0.83   4.53  -3.70   1.60
*    1.65  -0.65  -3.70   5.32  -1.37
*   -0.72   0.28   1.60  -1.37   1.98
*
*  and B is the right-hand side matrix:
*
*   -7.29   6.11   0.59
*    9.25   2.90   8.88
*    5.99  -5.05   7.57
*   -1.94  -3.80   5.57
*   -8.30   9.66  -1.67
*
*  Description.
*  ============
*
*  The routine solves for X the real system of linear equations
*  A*X = B, where A is an n-by-n symmetric positive-definite
*  matrix, the columns of matrix B are individual right-hand sides,
*  and the columns of X are the corresponding solutions.
*
*  The Cholesky decomposition is used to factor A as
*  A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
*  where U is an upper triangular matrix and L is a lower triangular matrix.
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* SPOSV Example Program Results
*
* Solution
*  -6.02   3.95  -3.14
*  15.62   4.32  13.05
```

```
*   3.02  -8.25   4.91
*   3.25  -4.83   6.11
*  -8.78   9.04  -3.57
*
* Details of Cholesky factorization
*   1.77   0.10  -0.51   0.93  -0.41
*   0.00   0.88   0.99  -0.84   0.36
*   0.00   0.00   1.81  -1.32   0.57
*   0.00   0.00   0.00   1.42   0.05
*   0.00   0.00   0.00   0.00   1.16
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      REAL             A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $  3.14, 0.00, 0.00, 0.00, 0.00,
     $  0.17, 0.79, 0.00, 0.00, 0.00,
     $ -0.90, 0.83, 4.53, 0.00, 0.00,
     $  1.65,-0.65,-3.70, 5.32, 0.00,
     $ -0.72, 0.28, 1.60,-1.37, 1.98
     $                   /
      DATA             B/
     $ -7.29, 9.25, 5.99,-1.94,-8.30,
     $  6.11, 2.90,-5.05,-3.80, 9.66,
     $  0.59, 8.88, 7.57, 5.57,-1.67
     $                   /
*
*     .. External Subroutines ..
      EXTERNAL         SPOSV
      EXTERNAL         PRINT_MATRIX
*
*     .. Executable Statements ..
      WRITE(*,*)'SPOSV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL SPOSV( 'Upper', N, NRHS, A, LDA, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The leading minor of order ',INFO,' is not positive'
         WRITE(*,*)'definite; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
```

```
*
*     Print details of Cholesky factorization.
*
      CALL PRINT_MATRIX( 'Details of Cholesky factorization', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of SPOSV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SPOSV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SPOSV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:
```

```
     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 SPOSV Example Program Results

 Solution
  -6.02   3.95  -3.14
  15.62   4.32  13.05
   3.02  -8.25   4.91
   3.25  -4.83   6.11
  -8.78   9.04  -3.57

 Details of Cholesky factorization
   1.77   0.10  -0.51   0.93  -0.41
   0.00   0.88   0.99  -0.84   0.36
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
#include <stdlib.h>
#include <stdio.h>

/* SPOSV prototype */
extern void sposv( char* uplo, int* n, int* nrhs, float* a, int* lda,
                float* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 5
#define NRHS 3
```

```
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        float a[LDA*N] = {
            3.14f,  0.00f,  0.00f,  0.00f,  0.00f,
            0.17f,  0.79f,  0.00f,  0.00f,  0.00f,
           -0.90f,  0.83f,  4.53f,  0.00f,  0.00f,
            1.65f, -0.65f, -3.70f,  5.32f,  0.00f,
           -0.72f,  0.28f,  1.60f, -1.37f,  1.98f
        };
        float b[LDB*NRHS] = {
           -7.29f,  9.25f,  5.99f, -1.94f, -8.30f,
            6.11f,  2.90f, -5.05f, -3.80f,  9.66f,
            0.59f,  8.88f,  7.57f,  5.57f, -1.67f
        };
        /* Executable statements */
        printf( " SPOSV Example Program Results\n" );
        /* Solve the equations A*X = B */
        sposv( "Upper", &n, &nrhs, a, &lda, b, &ldb, &info );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of SPOSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACK_sposv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
```

```
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   LAPACKE_sposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

LAPACKE_sposv (row-major, high-level) Example Program Results

Solution
 -6.02   3.95  -3.14
 15.62   4.32  13.05
  3.02  -8.25   4.91
  3.25  -4.83   6.11
 -8.78   9.04  -3.57

Details of Cholesky factorization
  1.77   0.10  -0.51   0.93  -0.41
  0.00   0.88   0.99  -0.84   0.36
```

```
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        float a[LDA*N] = {
            3.14f,  0.17f, -0.90f, 1.65f, -0.72f,
            0.00f,  0.79f,  0.83f, -0.65f,  0.28f,
            0.00f,  0.00f,  4.53f, -3.70f,  1.60f,
            0.00f,  0.00f,  0.00f, 5.32f, -1.37f,
            0.00f,  0.00f,  0.00f, 0.00f,  1.98f
        };
        float b[LDB*N] = {
           -7.29f,  6.11f,  0.59f,
            9.25f,  2.90f,  8.88f,
            5.99f, -5.05f,  7.57f,
           -1.94f, -3.80f,  5.57f,
           -8.30f,  9.66f, -1.67f
        };
        /* Executable statements */
        printf( "LAPACKE_sposv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sposv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_sposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_sposv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a symmetric positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

     3.14   0.17  -0.90   1.65  -0.72
     0.17   0.79   0.83  -0.65   0.28
    -0.90   0.83   4.53  -3.70   1.60
     1.65  -0.65  -3.70   5.32  -1.37
    -0.72   0.28   1.60  -1.37   1.98

   and B is the right-hand side matrix:

    -7.29   6.11   0.59
     9.25   2.90   8.88
     5.99  -5.05   7.57
    -1.94  -3.80   5.57
    -8.30   9.66  -1.67

   Description.
   ============

   The routine solves for X the real system of linear equations
   A*X = B, where A is an n-by-n symmetric positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UT*U, if uplo = 'U' or A = L*LT, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.
```

```
   Example Program Results.
   ========================

 LAPACKE_sposv (column-major, high-level) Example Program Results

 Solution
  -6.02   3.95  -3.14
  15.62   4.32  13.05
   3.02  -8.25   4.91
   3.25  -4.83   6.11
  -8.78   9.04  -3.57

 Details of Cholesky factorization
   1.77   0.10  -0.51   0.93  -0.41
   0.00   0.88   0.99  -0.84   0.36
   0.00   0.00   1.81  -1.32   0.57
   0.00   0.00   0.00   1.42   0.05
   0.00   0.00   0.00   0.00   1.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        float a[LDA*N] = {
            3.14f,  0.00f,  0.00f,  0.00f,  0.00f,
            0.17f,  0.79f,  0.00f,  0.00f,  0.00f,
           -0.90f,  0.83f,  4.53f,  0.00f,  0.00f,
            1.65f, -0.65f, -3.70f,  5.32f,  0.00f,
           -0.72f,  0.28f,  1.60f, -1.37f,  1.98f
        };
        float b[LDB*NRHS] = {
           -7.29f,  9.25f,  5.99f, -1.94f, -8.30f,
            6.11f,  2.90f, -5.05f, -3.80f,  9.66f,
            0.59f,  8.88f,  7.57f,  5.57f, -1.67f
        };
        /* Executable statements */
        printf( "LAPACKE_sposv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sposv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
```

```
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_sposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZPOSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square Hermitian positive-definite matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The Cholesky decomposition is used to factor *A* as:

$A = U^H * U$, if $uplo = 'U'$ or $A = L * L^H$, if $uplo = 'L'$

where

- *U* is an upper triangular matrix.
- *L* is a lower triangular matrix.

The factored form of *A* is then used to solve the system of equations $A * X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major[1] Data Layout

- Example program in C for Column Major[1] Data Layout

See Also

?posv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?posv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZPOSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZPOSV Example.
*  ==============
*
*  The program computes the solution to the system of linear
*  equations with a Hermitian positive-definite matrix A and multiple
*  right-hand sides B, where A is the coefficient matrix:
*
*  (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
*  (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
*  ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
*  ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)
*
*  and B is the right-hand side matrix:
*
*  ( -2.94,  5.79) (  8.44,  3.07)
*  (  8.12, -9.12) (  1.00, -4.62)
*  (  9.09, -5.03) (  3.64, -2.33)
*  (  7.36,  6.77) (  8.04,  2.87)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations
*  A*X = B, where A is an n-by-n Hermitian positive-definite
*  matrix, the columns of matrix B are individual right-hand sides,
*  and the columns of X are the corresponding solutions.
*
*  The Cholesky decomposition is used to factor A as
*  A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
*  where U is an upper triangular matrix and L is a lower triangular matrix.
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* ZPOSV Example Program Results
```

```
*
* Solution
* (  0.80,  1.62) (  2.52,  0.61)
* (  1.26, -1.78) (  0.01, -1.38)
* (  3.38, -0.29) (  2.42, -0.52)
* (  3.46,  2.92) (  3.77,  1.37)
*
* Details of Cholesky factorization
* (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
* (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
* ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
* ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
*
*     .. Local Scalars ..
      INTEGER          INFO
*
*     .. Local Arrays ..
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS )
      DATA             A/
     $ ( 5.96, 0.00),( 0.40, 1.19),(-0.83, 0.48),(-0.57,-0.40),
     $ ( 0.00, 0.00),( 7.95, 0.00),( 0.33,-0.09),( 0.22,-0.74),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 4.43, 0.00),(-1.09,-0.32),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 3.46, 0.00)
     $                 /
      DATA             B/
     $ (-2.94, 5.79),( 8.12,-9.12),( 9.09,-5.03),( 7.36, 6.77),
     $ ( 8.44, 3.07),( 1.00,-4.62),( 3.64,-2.33),( 8.04, 2.87)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZPOSV
      EXTERNAL         PRINT_MATRIX
*
*     .. Executable Statements ..
      WRITE(*,*)'ZPOSV Example Program Results'
*
*     Solve the equations A*X = B.
*
      CALL ZPOSV( 'Lower', N, NRHS, A, LDA, B, LDB, INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The leading minor of order ',INFO,' is not positive'
         WRITE(*,*)'definite; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
```

```
*
*     Print details of Cholesky factorization.
*
      CALL PRINT_MATRIX( 'Details of Cholesky factorization', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of ZPOSV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

ZPOSV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   ZPOSV Example.
   ==============

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:
```

```
   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 ZPOSV Example Program Results

 Solution
 (  0.80,  1.62) (  2.52,  0.61)
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZPOSV prototype */
extern void zposv( char* uplo, int* n, int* nrhs, dcomplex* a, int* lda,
                dcomplex* b, int* ldb, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );

/* Parameters */
#define N 4
#define NRHS 2
```

```
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        dcomplex a[LDA*N] = {
           { 5.96,  0.00}, { 0.40,  1.19}, {-0.83,  0.48}, {-0.57, -0.40},
           { 0.00,  0.00}, { 7.95,  0.00}, { 0.33, -0.09}, { 0.22, -0.74},
           { 0.00,  0.00}, { 0.00,  0.00}, { 4.43,  0.00}, {-1.09, -0.32},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 3.46,  0.00}
        };
        dcomplex b[LDB*NRHS] = {
           {-2.94,  5.79}, { 8.12, -9.12}, { 9.09, -5.03}, { 7.36,  6.77},
           { 8.44,  3.07}, { 1.00, -4.62}, { 3.64, -2.33}, { 8.04,  2.87}
        };
        /* Executable statements */
        printf( " ZPOSV Example Program Results\n" );
        /* Solve the equations A*X = B */
        zposv( "Lower", &n, &nrhs, a, &lda, b, &ldb, &info );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of ZPOSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_zposv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_zposv (row-major, high-level) Example Program Results

 Solution
 (  0.80,  1.62) (  2.52,  0.61)
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
#include <stdlib.h>
```

```
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex16 a[LDA*N] = {
           { 5.96,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           { 0.40,  1.19}, { 7.95,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-0.83,  0.48}, { 0.33, -0.09}, { 4.43,  0.00}, { 0.00,  0.00},
           {-0.57, -0.40}, { 0.22, -0.74}, {-1.09, -0.32}, { 3.46,  0.00}
        };
        MKL_Complex16 b[LDB*N] = {
           {-2.94,  5.79}, { 8.44,  3.07},
           { 8.12, -9.12}, { 1.00, -4.62},
           { 9.09, -5.03}, { 3.64, -2.33},
           { 7.36,  6.77}, { 8.04,  2.87}
        };
        /* Executable statements */
        printf( "LAPACKE_zposv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zposv( LAPACK_ROW_MAJOR, 'L', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

LAPACK_zposv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zposv Example.
   ======================

   The program computes the solution to the system of linear
   equations with a Hermitian positive-definite matrix A and multiple
   right-hand sides B, where A is the coefficient matrix:

   (  5.96,  0.00) (  0.40, -1.19) ( -0.83, -0.48) ( -0.57,  0.40)
   (  0.40,  1.19) (  7.95,  0.00) (  0.33,  0.09) (  0.22,  0.74)
   ( -0.83,  0.48) (  0.33, -0.09) (  4.43,  0.00) ( -1.09,  0.32)
   ( -0.57, -0.40) (  0.22, -0.74) ( -1.09, -0.32) (  3.46,  0.00)

   and B is the right-hand side matrix:

   ( -2.94,  5.79) (  8.44,  3.07)
   (  8.12, -9.12) (  1.00, -4.62)
   (  9.09, -5.03) (  3.64, -2.33)
   (  7.36,  6.77) (  8.04,  2.87)

   Description.
   ============

   The routine solves for X the complex system of linear equations
   A*X = B, where A is an n-by-n Hermitian positive-definite
   matrix, the columns of matrix B are individual right-hand sides,
   and the columns of X are the corresponding solutions.

   The Cholesky decomposition is used to factor A as
   A = UH*U, if uplo = 'U' or A = L*LH, if uplo = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.
   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_zposv (column-major, high-level) Example Program Results

 Solution
 (  0.80,  1.62) (  2.52,  0.61)
```

```
 (  1.26, -1.78) (  0.01, -1.38)
 (  3.38, -0.29) (  2.42, -0.52)
 (  3.46,  2.92) (  3.77,  1.37)

 Details of Cholesky factorization
 (  2.44,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 (  0.16,  0.49) (  2.77,  0.00) (  0.00,  0.00) (  0.00,  0.00)
 ( -0.34,  0.20) (  0.10, -0.10) (  2.06,  0.00) (  0.00,  0.00)
 ( -0.23, -0.16) (  0.12, -0.30) ( -0.57, -0.20) (  1.71,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex16 a[LDA*N] = {
           { 5.96,  0.00}, { 0.40,  1.19}, {-0.83,  0.48}, {-0.57, -0.40},
           { 0.00,  0.00}, { 7.95,  0.00}, { 0.33, -0.09}, { 0.22, -0.74},
           { 0.00,  0.00}, { 0.00,  0.00}, { 4.43,  0.00}, {-1.09, -0.32},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 3.46,  0.00}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           {-2.94,  5.79}, { 8.12, -9.12}, { 9.09, -5.03}, { 7.36,  6.77},
           { 8.44,  3.07}, { 1.00, -4.62}, { 3.64, -2.33}, { 8.04,  2.87}
        };
        /* Executable statements */
        printf( "LAPACKE_zposv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zposv( LAPACK_COL_MAJOR, 'L', n, nrhs, a, lda, b, ldb );
        /* Check for the positive definiteness */
        if( info > 0 ) {
                printf( "The leading minor of order %i is not positive ", info );
                printf( "definite;\nthe solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of Cholesky factorization */
        print_matrix( "Details of Cholesky factorization", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zposv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

?sysv function

Computes the solution to the system of linear equations with a real or complex symmetric coefficient matrix *A* and multiple right-hand sides.

NOTE
For a detailed description and reference information on this function, please visit:

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call ssysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call dsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call csysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

Calling from C:

```
ssysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
dsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
csysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
zsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
```

C Interface

```
lapack_int LAPACKE_ssysv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
float* a, lapack_int lda, lapack_int* ipiv, float* b, lapack_int ldb);
lapack_int LAPACKE_dsysv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
double* a, lapack_int lda, lapack_int* ipiv, double* b, lapack_int ldb);
lapack_int LAPACKE_csysv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_float* a, lapack_int lda, lapack_int* ipiv, lapack_complex_float* b,
lapack_int ldb);
lapack_int LAPACKE_zsysv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_double* a, lapack_int lda, lapack_int* ipiv, lapack_complex_double* b,
lapack_int ldb);
```

See Also

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CSYSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^T$ or $A = L*D*L^T$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a symmetric and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CSYSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
```

```
*  in writing.
*  ============================================================================
*
*  CSYSV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a complex symmetric matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*  (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
*  ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
*  ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
*  ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)
*
*  and B is the right-hand side matrix:
*
*  (  5.71, -1.20) (  2.84, -0.18)
*  ( -7.70,  6.47) ( -8.29, -1.72)
*  (  3.77, -7.40) ( -4.28, -8.25)
*  ( -3.78,  0.33) ( -2.70, -0.39)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations A*X = B,
*  where A is an n-by-n symmetric matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The diagonal pivoting method is used to factor A as A = U*D*UT or
*  A = L*D*LT , where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is symmetric and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* CSYSV Example Program Results
*
* Solution
* (  0.13,  0.13) (  0.63,  0.34)
* (  0.32, -0.07) (  0.61,  0.21)
* ( -0.26, -0.44) ( -0.01, -0.10)
* ( -0.40,  0.51) (  0.21,  0.02)
*
* Details of factorization
* (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
* (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
* (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
* (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)
*
* Pivot indices
*      1     -1     -1      3
*  ============================================================================
*
```

```
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX          A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ ( 9.99,-4.73),( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ (-5.68,-0.80),(-8.01, 4.61),( 0.00, 0.00),( 0.00, 0.00),
     $ (-8.94, 1.32),( 1.64,-6.29),( 9.04, 3.96),( 0.00, 0.00),
     $ (-9.42, 2.05),( 6.79,-2.17),(-4.51,-7.54),( 0.40, 4.06)
     $                 /
      DATA             B/
     $ ( 5.71,-1.20),(-7.70, 6.47),( 3.77,-7.40),(-3.78, 0.33),
     $ ( 2.84,-0.18),(-8.29,-1.72),(-4.28,-8.25),(-2.70,-0.39)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CSYSV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CSYSV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CSYSV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL CSYSV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
```

```
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of CSYSV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

CSYSV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
```

```
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   CSYSV Example.
   ==============

   The program computes the solution to the system of linear equations
   with a complex symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
   ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
   ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
   ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

   and B is the right-hand side matrix:

   (  5.71, -1.20) (  2.84, -0.18)
   ( -7.70,  6.47) ( -8.29, -1.72)
   (  3.77, -7.40) ( -4.28, -8.25)
   ( -3.78,  0.33) ( -2.70, -0.39)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 CSYSV Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)
```

```
 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CSYSV prototype */
extern void csysv( char* uplo, int* n, int* nrhs, fcomplex* a, int* lda,
                int* ipiv, fcomplex* b, int* ldb, fcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        int ipiv[N];
        fcomplex a[LDA*N] = {
           { 9.99f, -4.73f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-5.68f, -0.80f}, {-8.01f,  4.61f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-8.94f,  1.32f}, { 1.64f, -6.29f}, { 9.04f,  3.96f}, { 0.00f,  0.00f},
           {-9.42f,  2.05f}, { 6.79f, -2.17f}, {-4.51f, -7.54f}, { 0.40f,  4.06f}
        };
        fcomplex b[LDB*NRHS] = {
           { 5.71f, -1.20f}, {-7.70f,  6.47f}, { 3.77f, -7.40f}, {-3.78f,  0.33f},
           { 2.84f, -0.18f}, {-8.29f, -1.72f}, {-4.28f, -8.25f}, {-2.70f, -0.39f}
        };
        /* Executable statements */
        printf( " CSYSV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        csysv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve the equations A*X = B */
        csysv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
```

```
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CSYSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_csysv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_csysv Example.
   ======================
```

```
  The program computes the solution to the system of linear equations
  with a complex symmetric matrix A and multiple right-hand sides B,
  where A is the coefficient matrix:

  (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
  ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
  ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
  ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

  and B is the right-hand side matrix:

  (  5.71, -1.20) (  2.84, -0.18)
  ( -7.70,  6.47) ( -8.29, -1.72)
  (  3.77, -7.40) ( -4.28, -8.25)
  ( -3.78,  0.33) ( -2.70, -0.39)

  Description.
  ============

  The routine solves for X the complex system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_csysv (row-major, high-level) Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)

 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
```

```
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           { 9.99f, -4.73f}, {-5.68f, -0.80f}, {-8.94f,  1.32f}, {-9.42f,  2.05f},
           { 0.00f,  0.00f}, {-8.01f,  4.61f}, { 1.64f, -6.29f}, { 6.79f, -2.17f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 9.04f,  3.96f}, {-4.51f, -7.54f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.40f,  4.06f}
        };
        MKL_Complex8 b[LDB*N] = {
           { 5.71f, -1.20f}, { 2.84f, -0.18f},
           {-7.70f,  6.47f}, {-8.29f, -1.72f},
           { 3.77f, -7.40f}, {-4.28f, -8.25f},
           {-3.78f,  0.33f}, {-2.70f, -0.39f}
        };
        /* Executable statements */
        printf( "LAPACKE_csysv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_csysv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_csysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

```
/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_csysv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_csysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a complex symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
   ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
   ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
   ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

   and B is the right-hand side matrix:

   (  5.71, -1.20) (  2.84, -0.18)
   ( -7.70,  6.47) ( -8.29, -1.72)
   (  3.77, -7.40) ( -4.28, -8.25)
   ( -3.78,  0.33) ( -2.70, -0.39)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
```

```
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_csysv (column-major, high-level) Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)

 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           { 9.99f, -4.73f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-5.68f, -0.80f}, {-8.01f,  4.61f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-8.94f,  1.32f}, { 1.64f, -6.29f}, { 9.04f,  3.96f}, { 0.00f,  0.00f},
           {-9.42f,  2.05f}, { 6.79f, -2.17f}, {-4.51f, -7.54f}, { 0.40f,  4.06f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           { 5.71f, -1.20f}, {-7.70f,  6.47f}, { 3.77f, -7.40f}, {-3.78f,  0.33f},
           { 2.84f, -0.18f}, {-8.29f, -1.72f}, {-4.28f, -8.25f}, {-2.70f, -0.39f}
        };
        /* Executable statements */
        printf( "LAPACKE_csysv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_csysv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, ipiv,
```

```
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_csysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

DSYSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the real system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^T$ or $A = L*D*L^T$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a symmetric and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major[1] Data Layout
- Example program in C for Column Major[1] Data Layout

See Also

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DSYSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DSYSV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a real symmetric matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*   -5.86   3.99  -5.93  -2.82   7.69
*    3.99   4.46   2.58   4.42   4.61
*   -5.93   2.58  -8.52   8.57   7.69
*   -2.82   4.42   8.57   3.72   8.07
*    7.69   4.61   7.69   8.07   9.83
*
*  and B is the right-hand side matrix:
*
*    1.32  -6.33  -8.77
*    2.22   1.69  -8.33
*    0.12  -1.56   9.54
*   -6.41  -9.49   9.56
*    6.33  -3.67   7.48
```

```
*
*  Description.
*  ============
*
*  The routine solves for X the real system of linear equations A*X = B,
*  where A is an n-by-n symmetric matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The diagonal pivoting method is used to factor A as A = U*D*UT or
*  A = L*D*LT , where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is symmetric and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* DSYSV Example Program Results
*
* Solution
*   1.17   0.52  -0.86
*  -0.71   1.05  -4.90
*  -0.63  -0.52   0.99
*  -0.33   0.43   1.22
*   0.83  -1.22   1.96
*
* Details of factorization
*  -5.86   0.00   0.00   0.00   0.00
*  -0.68   7.18   0.00   0.00   0.00
*   1.01  -0.20  -2.82   0.00   0.00
*   0.48   0.35  11.93   4.21   0.00
*  -1.31   1.37   0.02   0.16   6.22
*
* Pivot indices
*      1      2     -4     -4      5
*  ==============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ -5.86, 3.99,-5.93,-2.82, 7.69,
     $  0.00, 4.46, 2.58, 4.42, 4.61,
     $  0.00, 0.00,-8.52, 8.57, 7.69,
     $  0.00, 0.00, 0.00, 3.72, 8.07,
```

```
     $  0.00, 0.00, 0.00, 0.00, 9.83
     $                  /
      DATA             B/
     $  1.32, 2.22, 0.12,-6.41, 6.33,
     $ -6.33, 1.69,-1.56,-9.49,-3.67,
     $ -8.77,-8.33, 9.54, 9.56, 7.48
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DSYSV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DSYSV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DSYSV( 'Lower', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL DSYSV( 'Lower', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of DSYSV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
```

```
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

DSYSV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   DSYSV Example.
   ==============
```

```
  The program computes the solution to the system of linear equations
  with a real symmetric matrix A and multiple right-hand sides B,
  where A is the coefficient matrix:

   -5.86   3.99  -5.93  -2.82   7.69
    3.99   4.46   2.58   4.42   4.61
   -5.93   2.58  -8.52   8.57   7.69
   -2.82   4.42   8.57   3.72   8.07
    7.69   4.61   7.69   8.07   9.83

  and B is the right-hand side matrix:

    1.32  -6.33  -8.77
    2.22   1.69  -8.33
    0.12  -1.56   9.54
   -6.41  -9.49   9.56
    6.33  -3.67   7.48

  Description.
  ============

  The routine solves for X the real system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 DSYSV Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>
```

```
/* DSYSV prototype */
extern void dsysv( char* uplo, int* n, int* nrhs, double* a, int* lda,
                int* ipiv, double* b, int* ldb, double* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        double wkopt;
        double* work;
        /* Local arrays */
        int ipiv[N];
        double a[LDA*N] = {
           -5.86,  3.99, -5.93, -2.82,  7.69,
            0.00,  4.46,  2.58,  4.42,  4.61,
            0.00,  0.00, -8.52,  8.57,  7.69,
            0.00,  0.00,  0.00,  3.72,  8.07,
            0.00,  0.00,  0.00,  0.00,  9.83
        };
        double b[LDB*NRHS] = {
            1.32,  2.22,  0.12, -6.41,  6.33,
           -6.33,  1.69, -1.56, -9.49, -3.67,
           -8.77, -8.33,  9.54,  9.56,  7.48
        };
        /* Executable statements */
        printf( " DSYSV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dsysv( "Lower", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve the equations A*X = B */
        dsysv( "Lower", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
```

```
} /* End of DSYSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_dsysv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a real symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

    -5.86   3.99  -5.93  -2.82   7.69
     3.99   4.46   2.58   4.42   4.61
    -5.93   2.58  -8.52   8.57   7.69
    -2.82   4.42   8.57   3.72   8.07
     7.69   4.61   7.69   8.07   9.83

   and B is the right-hand side matrix:

     1.32  -6.33  -8.77
     2.22   1.69  -8.33
     0.12  -1.56   9.54
```

```
    -6.41  -9.49   9.56
     6.33  -3.67   7.48

  Description.
  ============

  The routine solves for X the real system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_dsysv (row-major, high-level) Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
```

```
        /* Local arrays */
        MKL_INT ipiv[N];
        double a[LDA*N] = {
           -5.86,  0.00,  0.00,  0.00,  0.00,
            3.99,  4.46,  0.00,  0.00,  0.00,
           -5.93,  2.58, -8.52,  0.00,  0.00,
           -2.82,  4.42,  8.57,  3.72,  0.00,
            7.69,  4.61,  7.69,  8.07,  9.83
        };
        double b[LDB*N] = {
            1.32, -6.33, -8.77,
            2.22,  1.69, -8.33,
            0.12, -1.56, 9.54,
           -6.41, -9.49, 9.56,
            6.33, -3.67, 7.48
        };
        /* Executable statements */
        printf( "LAPACKE_dsysv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dsysv( LAPACK_ROW_MAJOR, 'L', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_dsysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_dsysv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a real symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

    -5.86   3.99  -5.93  -2.82   7.69
     3.99   4.46   2.58   4.42   4.61
    -5.93   2.58  -8.52   8.57   7.69
    -2.82   4.42   8.57   3.72   8.07
     7.69   4.61   7.69   8.07   9.83

   and B is the right-hand side matrix:

     1.32  -6.33  -8.77
     2.22   1.69  -8.33
     0.12  -1.56   9.54
    -6.41  -9.49   9.56
     6.33  -3.67   7.48

   Description.
   ============

   The routine solves for X the real system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================
```

```
 LAPACKE_dsysv (column-major, high-level) Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        double a[LDA*N] = {
           -5.86,  3.99, -5.93, -2.82,  7.69,
            0.00,  4.46,  2.58,  4.42,  4.61,
            0.00,  0.00, -8.52,  8.57,  7.69,
            0.00,  0.00,  0.00,  3.72,  8.07,
            0.00,  0.00,  0.00,  0.00,  9.83
        };
        double b[LDB*NRHS] = {
            1.32,  2.22,  0.12, -6.41,  6.33,
           -6.33,  1.69, -1.56, -9.49, -3.67,
           -8.77, -8.33,  9.54,  9.56,  7.48
        };
        /* Executable statements */
        printf( "LAPACKE_dsysv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dsysv( LAPACK_COL_MAJOR, 'L', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
```

```
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_dsysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

SSYSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the real system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^T$ or $A = L*D*L^T$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a symmetric and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations `A*X = B`.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SSYSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SSYSV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a real symmetric matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*   -5.86   3.99  -5.93  -2.82   7.69
*    3.99   4.46   2.58   4.42   4.61
*   -5.93   2.58  -8.52   8.57   7.69
*   -2.82   4.42   8.57   3.72   8.07
*    7.69   4.61   7.69   8.07   9.83
*
*  and B is the right-hand side matrix:
*
*    1.32  -6.33  -8.77
*    2.22   1.69  -8.33
*    0.12  -1.56   9.54
*   -6.41  -9.49   9.56
*    6.33  -3.67   7.48
*
*  Description.
*  ============
```

```
*
*  The routine solves for X the real system of linear equations A*X = B,
*  where A is an n-by-n symmetric matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The diagonal pivoting method is used to factor A as A = U*D*UT or
*  A = L*D*LT , where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is symmetric and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* SSYSV Example Program Results
*
* Solution
*   1.17   0.52  -0.86
*  -0.71   1.05  -4.90
*  -0.63  -0.52   0.99
*  -0.33   0.43   1.22
*   0.83  -1.22   1.96
*
* Details of factorization
*  -5.86   0.00   0.00   0.00   0.00
*  -0.68   7.18   0.00   0.00   0.00
*   1.01  -0.20  -2.82   0.00   0.00
*   0.48   0.35  11.93   4.21   0.00
*  -1.31   1.37   0.02   0.16   6.22
*
* Pivot indices
*      1      2     -4     -4      5
*  ==============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      REAL             A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ -5.86, 3.99,-5.93,-2.82, 7.69,
     $  0.00, 4.46, 2.58, 4.42, 4.61,
     $  0.00, 0.00,-8.52, 8.57, 7.69,
     $  0.00, 0.00, 0.00, 3.72, 8.07,
     $  0.00, 0.00, 0.00, 0.00, 9.83
     $                  /
      DATA             B/
```

```
     $  1.32, 2.22, 0.12,-6.41, 6.33,
     $ -6.33, 1.69,-1.56,-9.49,-3.67,
     $ -8.77,-8.33, 9.54, 9.56, 7.48
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SSYSV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SSYSV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SSYSV( 'Lower', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL SSYSV( 'Lower', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of SSYSV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
```

```
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

SYSV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SSYSV Example.
   ==============

   The program computes the solution to the system of linear equations
   with a real symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:
```

```
   -5.86   3.99  -5.93  -2.82   7.69
    3.99   4.46   2.58   4.42   4.61
   -5.93   2.58  -8.52   8.57   7.69
   -2.82   4.42   8.57   3.72   8.07
    7.69   4.61   7.69   8.07   9.83

  and B is the right-hand side matrix:

    1.32  -6.33  -8.77
    2.22   1.69  -8.33
    0.12  -1.56   9.54
   -6.41  -9.49   9.56
    6.33  -3.67   7.48

  Description.
  ============

  The routine solves for X the real system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 SSYSV Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>

/* SSYSV prototype */
extern void ssysv( char* uplo, int* n, int* nrhs, float* a, int* lda,
                int* ipiv, float* b, int* ldb, float* work, int* lwork, int* info );
```

```
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
        int ipiv[N];
        float a[LDA*N] = {
           -5.86f,  3.99f, -5.93f, -2.82f,  7.69f,
            0.00f,  4.46f,  2.58f,  4.42f,  4.61f,
            0.00f,  0.00f, -8.52f,  8.57f,  7.69f,
            0.00f,  0.00f,  0.00f,  3.72f,  8.07f,
            0.00f,  0.00f,  0.00f,  0.00f,  9.83f
        };
        float b[LDB*NRHS] = {
            1.32f,  2.22f,  0.12f, -6.41f,  6.33f,
           -6.33f,  1.69f, -1.56f, -9.49f, -3.67f,
           -8.77f, -8.33f,  9.54f,  9.56f,  7.48f
        };
        /* Executable statements */
        printf( " SSYSV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        ssysv( "Lower", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve the equations A*X = B */
        ssysv( "Lower", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SSYSV Example */

/* Auxiliary routine: printing a matrix */
```

```
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_ssysv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a real symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

    -5.86   3.99  -5.93  -2.82   7.69
     3.99   4.46   2.58   4.42   4.61
    -5.93   2.58  -8.52   8.57   7.69
    -2.82   4.42   8.57   3.72   8.07
     7.69   4.61   7.69   8.07   9.83

   and B is the right-hand side matrix:

     1.32  -6.33  -8.77
     2.22   1.69  -8.33
     0.12  -1.56   9.54
    -6.41  -9.49   9.56
     6.33  -3.67   7.48
```

```
  Description.
  ============

  The routine solves for X the real system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_ssysv (row-major, high-level) Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        float a[LDA*N] = {
```

```
           -5.86f,  0.00f,  0.00f,  0.00f,  0.00f,
            3.99f,  4.46f,  0.00f,  0.00f,  0.00f,
           -5.93f,  2.58f, -8.52f,  0.00f,  0.00f,
           -2.82f,  4.42f,  8.57f,  3.72f,  0.00f,
            7.69f,  4.61f,  7.69f,  8.07f,  9.83f
        };
        float b[LDB*N] = {
            1.32f, -6.33f, -8.77f,
            2.22f,  1.69f, -8.33f,
            0.12f, -1.56f, 9.54f,
           -6.41f, -9.49f, 9.56f,
            6.33f, -3.67f, 7.48f
        };
        /* Executable statements */
        printf( "LAPACKE_ssysv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_ssysv( LAPACK_ROW_MAJOR, 'L', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_ssysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_ssysv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
```

```
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   LAPACKE_ssysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a real symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

    -5.86   3.99  -5.93  -2.82   7.69
     3.99   4.46   2.58   4.42   4.61
    -5.93   2.58  -8.52   8.57   7.69
    -2.82   4.42   8.57   3.72   8.07
     7.69   4.61   7.69   8.07   9.83

   and B is the right-hand side matrix:

     1.32  -6.33  -8.77
     2.22   1.69  -8.33
     0.12  -1.56   9.54
    -6.41  -9.49   9.56
     6.33  -3.67   7.48

   Description.
   ============

   The routine solves for X the real system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_ssysv (column-major, high-level) Example Program Results

 Solution
   1.17   0.52  -0.86
  -0.71   1.05  -4.90
```

```
  -0.63  -0.52   0.99
  -0.33   0.43   1.22
   0.83  -1.22   1.96

 Details of factorization
  -5.86   0.00   0.00   0.00   0.00
  -0.68   7.18   0.00   0.00   0.00
   1.01  -0.20  -2.82   0.00   0.00
   0.48   0.35  11.93   4.21   0.00
  -1.31   1.37   0.02   0.16   6.22

 Pivot indices
      1      2     -4     -4      5
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 5
#define NRHS 3
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        float a[LDA*N] = {
           -5.86f,  3.99f, -5.93f, -2.82f,  7.69f,
            0.00f,  4.46f,  2.58f,  4.42f,  4.61f,
            0.00f,  0.00f, -8.52f,  8.57f,  7.69f,
            0.00f,  0.00f,  0.00f,  3.72f,  8.07f,
            0.00f,  0.00f,  0.00f,  0.00f,  9.83f
        };
        float b[LDB*NRHS] = {
            1.32f,  2.22f,  0.12f, -6.41f,  6.33f,
           -6.33f,  1.69f, -1.56f, -9.49f, -3.67f,
           -8.77f, -8.33f,  9.54f,  9.56f,  7.48f
        };
        /* Executable statements */
        printf( "LAPACKE_ssysv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_ssysv( LAPACK_COL_MAJOR, 'L', n, nrhs, a, lda, ipiv,
                              b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
```

```
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_ssysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

ZSYSV Example

NOTE
For a detailed description and reference information on this function, please visit:

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square symmetric matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^T$ or $A = L*D*L^T$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a symmetric and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?sysv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZSYSV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZSYSV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a complex symmetric matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*  (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
*  ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
*  ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
*  ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)
*
*  and B is the right-hand side matrix:
*
*  (  5.71, -1.20) (  2.84, -0.18)
*  ( -7.70,  6.47) ( -8.29, -1.72)
*  (  3.77, -7.40) ( -4.28, -8.25)
*  ( -3.78,  0.33) ( -2.70, -0.39)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations A*X = B,
*  where A is an n-by-n symmetric matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
```

```
*
*  The diagonal pivoting method is used to factor A as A = U*D*UT or
*  A = L*D*LT , where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is symmetric and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* ZSYSV Example Program Results
*
* Solution
* (  0.13,  0.13) (  0.63,  0.34)
* (  0.32, -0.07) (  0.61,  0.21)
* ( -0.26, -0.44) ( -0.01, -0.10)
* ( -0.40,  0.51) (  0.21,  0.02)
*
* Details of factorization
* (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
* (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
* (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
* (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)
*
* Pivot indices
*      1     -1     -1      3
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ ( 9.99,-4.73),( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ (-5.68,-0.80),(-8.01, 4.61),( 0.00, 0.00),( 0.00, 0.00),
     $ (-8.94, 1.32),( 1.64,-6.29),( 9.04, 3.96),( 0.00, 0.00),
     $ (-9.42, 2.05),( 6.79,-2.17),(-4.51,-7.54),( 0.40, 4.06)
     $                 /
      DATA             B/
     $ ( 5.71,-1.20),(-7.70, 6.47),( 3.77,-7.40),(-3.78, 0.33),
     $ ( 2.84,-0.18),(-8.29,-1.72),(-4.28,-8.25),(-2.70,-0.39)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZSYSV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZSYSV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZSYSV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL ZSYSV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of ZSYSV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
```

```
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

ZSYSV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   ZSYSV Example.
   ==============

   The program computes the solution to the system of linear equations
   with a complex symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
   ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
   ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
   ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

   and B is the right-hand side matrix:

   (  5.71, -1.20) (  2.84, -0.18)
```

```
   ( -7.70,  6.47) ( -8.29, -1.72)
   (  3.77, -7.40) ( -4.28, -8.25)
   ( -3.78,  0.33) ( -2.70, -0.39)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 ZSYSV Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)

 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZSYSV prototype */
extern void zsysv( char* uplo, int* n, int* nrhs, dcomplex* a, int* lda,
                int* ipiv, dcomplex* b, int* ldb, dcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N
```

```
/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        int ipiv[N];
        dcomplex a[LDA*N] = {
           { 9.99, -4.73}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-5.68, -0.80}, {-8.01,  4.61}, { 0.00,  0.00}, { 0.00,  0.00},
           {-8.94,  1.32}, { 1.64, -6.29}, { 9.04,  3.96}, { 0.00,  0.00},
           {-9.42,  2.05}, { 6.79, -2.17}, {-4.51, -7.54}, { 0.40,  4.06}
        };
        dcomplex b[LDB*NRHS] = {
           { 5.71, -1.20}, {-7.70,  6.47}, { 3.77, -7.40}, {-3.78,  0.33},
           { 2.84, -0.18}, {-8.29, -1.72}, {-4.28, -8.25}, {-2.70, -0.39}
        };
        /* Executable statements */
        printf( " ZSYSV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zsysv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve the equations A*X = B */
        zsysv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZSYSV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
```

```
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zsysv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zsysv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a complex symmetric matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
   ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
   ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
   ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

   and B is the right-hand side matrix:

   (  5.71, -1.20) (  2.84, -0.18)
   ( -7.70,  6.47) ( -8.29, -1.72)
   (  3.77, -7.40) ( -4.28, -8.25)
   ( -3.78,  0.33) ( -2.70, -0.39)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n symmetric matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UT or
   A = L*D*LT , where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.
```

```
  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_zsysv (row-major, high-level) Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)

 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           { 9.99, -4.73}, {-5.68, -0.80}, {-8.94,  1.32}, {-9.42,  2.05},
           { 0.00,  0.00}, {-8.01,  4.61}, { 1.64, -6.29}, { 6.79, -2.17},
           { 0.00,  0.00}, { 0.00,  0.00}, { 9.04,  3.96}, {-4.51, -7.54},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.40,  4.06}
        };
        MKL_Complex16 b[LDB*N] = {
           { 5.71, -1.20}, { 2.84, -0.18},
           {-7.70,  6.47}, {-8.29, -1.72},
           { 3.77, -7.40}, {-4.28, -8.25},
           {-3.78,  0.33}, {-2.70, -0.39}
        };
        /* Executable statements */
        printf( "LAPACKE_zsysv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
```

```
        info = LAPACKE_zsysv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zsysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zsysv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zsysv Example.
```

```
  ======================

  The program computes the solution to the system of linear equations
  with a complex symmetric matrix A and multiple right-hand sides B,
  where A is the coefficient matrix:

  (  9.99, -4.73) ( -5.68, -0.80) ( -8.94,  1.32) ( -9.42,  2.05)
  ( -5.68, -0.80) ( -8.01,  4.61) (  1.64, -6.29) (  6.79, -2.17)
  ( -8.94,  1.32) (  1.64, -6.29) (  9.04,  3.96) ( -4.51, -7.54)
  ( -9.42,  2.05) (  6.79, -2.17) ( -4.51, -7.54) (  0.40,  4.06)

  and B is the right-hand side matrix:

  (  5.71, -1.20) (  2.84, -0.18)
  ( -7.70,  6.47) ( -8.29, -1.72)
  (  3.77, -7.40) ( -4.28, -8.25)
  ( -3.78,  0.33) ( -2.70, -0.39)

  Description.
  ============

  The routine solves for X the complex system of linear equations A*X = B,
  where A is an n-by-n symmetric matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UT or
  A = L*D*LT , where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is symmetric and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_zsysv (column-major, high-level) Example Program Results

 Solution
 (  0.13,  0.13) (  0.63,  0.34)
 (  0.32, -0.07) (  0.61,  0.21)
 ( -0.26, -0.44) ( -0.01, -0.10)
 ( -0.40,  0.51) (  0.21,  0.02)

 Details of factorization
 (-16.42,  1.69) ( -0.53,  0.35) (  0.36,  0.41) ( -0.78,  0.49)
 (  0.00,  0.00) (  3.69,  0.64) (-16.58, -1.61) ( -0.10, -0.65)
 (  0.00,  0.00) (  0.00,  0.00) (  1.02, -3.74) ( -0.73, -0.52)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  9.04,  3.96)

 Pivot indices
      1     -1     -1      3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
```

```
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           { 9.99, -4.73}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-5.68, -0.80}, {-8.01,  4.61}, { 0.00,  0.00}, { 0.00,  0.00},
           {-8.94,  1.32}, { 1.64, -6.29}, { 9.04,  3.96}, { 0.00,  0.00},
           {-9.42,  2.05}, { 6.79, -2.17}, {-4.51, -7.54}, { 0.40,  4.06}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           { 5.71, -1.20}, {-7.70,  6.47}, { 3.77, -7.40}, {-3.78,  0.33},
           { 2.84, -0.18}, {-8.29, -1.72}, {-4.28, -8.25}, {-2.70, -0.39}
        };
        /* Executable statements */
        printf( "LAPACKE_zsysv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zsysv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zsysv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
```

```
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

?hesv function

Computes the solution to the system of linear equations with a Hermitian matrix *A* and multiple right-hand sides.

NOTE
For a detailed description and reference information on this function, please visit:

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call chesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zhesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

Calling from C:

```
chesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
zhesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info);
```

C Interface

```
lapack_int LAPACKE_chesv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_float* a, lapack_int lda, lapack_int* ipiv, lapack_complex_float* b,
lapack_int ldb);
lapack_int LAPACKE_zhesv(int matrix_layout, char uplo, lapack_int n, lapack_int nrhs,
lapack_complex_double* a, lapack_int lda, lapack_int* ipiv, lapack_complex_double* b,
lapack_int ldb);
```

See Also

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CHESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square Hermitian matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^H$ or $A = L*D*L^H$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a Hermitian and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CHESV Example Program in Fortran

```
Intel® oneAPI Math Kernel Library LAPACK Examples

*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  ================================================================================
*
*  CHESV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a Hermitian matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*  ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
```

```
*  (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
*  (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
*  ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)
*
*  and B is the right-hand side matrix:
*
*  ( -5.69, -8.21) ( -2.83,  6.46)
*  ( -3.57,  1.99) ( -7.64,  1.10)
*  (  8.42, -9.83) ( -2.33, -4.23)
*  ( -5.00,  3.85) (  6.48, -3.81)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations A*X = B,
*  where A is an n-by-n Hermitian matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
*
*  The diagonal pivoting method is used to factor A as A = U*D*UH or
*  A = L*D*LH, where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is Hermitian and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* CHESV Example Program Results
*
* Solution
* (  0.22, -0.95) ( -1.13,  0.18)
* ( -1.42, -1.30) (  0.70,  1.13)
* ( -0.65, -0.40) (  0.04,  0.07)
* ( -0.48,  1.35) (  1.15, -0.27)
*
* Details of factorization
* (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
* (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
* (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
* (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)
*
* Pivot indices
*     -1     -1     -3     -3
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
```

```
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX          A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ (-2.90, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ ( 0.31, 4.46),(-7.93, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ ( 9.66,-5.66),( 9.55,-4.62),( 0.30, 0.00),( 0.00, 0.00),
     $ (-2.28, 2.14),(-3.51, 3.11),( 9.33,-9.66),( 2.40, 0.00)
     $                 /
      DATA             B/
     $ (-5.69,-8.21),(-3.57, 1.99),( 8.42,-9.83),(-5.00, 3.85),
     $ (-2.83, 6.46),(-7.64, 1.10),(-2.33,-4.23),( 6.48,-3.81)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CHESV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CHESV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CHESV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL CHESV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
```

```
*
*     End of CHESV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

CHESV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
```

```
*****************************************************************************
*/
/*
   CHESV Example.
   ==============

   The program computes the solution to the system of linear equations
   with a Hermitian matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
   (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
   (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
   ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

   and B is the right-hand side matrix:

   ( -5.69, -8.21) ( -2.83,  6.46)
   ( -3.57,  1.99) ( -7.64,  1.10)
   (  8.42, -9.83) ( -2.33, -4.23)
   ( -5.00,  3.85) (  6.48, -3.81)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n Hermitian matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UH or
   A = L*D*LH, where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is Hermitian and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 CHESV Example Program Results

 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
```

```
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CHESV prototype */
extern void chesv( char* uplo, int* n, int* nrhs, fcomplex* a, int* lda,
                int* ipiv, fcomplex* b, int* ldb, fcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        int ipiv[N];
        fcomplex a[LDA*N] = {
           {-2.90f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           { 0.31f,  4.46f}, {-7.93f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           { 9.66f, -5.66f}, { 9.55f, -4.62f}, { 0.30f,  0.00f}, { 0.00f,  0.00f},
           {-2.28f,  2.14f}, {-3.51f,  3.11f}, { 9.33f, -9.66f}, { 2.40f,  0.00f}
        };
        fcomplex b[LDB*NRHS] = {
           {-5.69f, -8.21f}, {-3.57f,  1.99f}, { 8.42f, -9.83f}, {-5.00f,  3.85f},
           {-2.83f,  6.46f}, {-7.64f,  1.10f}, {-2.33f, -4.23f}, { 6.48f, -3.81f}
        };
        /* Executable statements */
        printf( " CHESV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        chesv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve the equations A*X = B */
        chesv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
```

```
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CHESV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_chesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_chesv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a Hermitian matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
   (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
   (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
   ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

   and B is the right-hand side matrix:
```

```
   ( -5.69, -8.21) ( -2.83,  6.46)
   ( -3.57,  1.99) ( -7.64,  1.10)
   (  8.42, -9.83) ( -2.33, -4.23)
   ( -5.00,  3.85) (  6.48, -3.81)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n Hermitian matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UH or
   A = L*D*LH, where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is Hermitian and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_chesv (row-major, high-level) Example Program Results

 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
```

```
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           {-2.90f,  0.00f}, { 0.31f,  4.46f}, { 9.66f, -5.66f}, {-2.28f,  2.14f},
           { 0.00f,  0.00f}, {-7.93f,  0.00f}, { 9.55f, -4.62f}, {-3.51f,  3.11f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.30f,  0.00f}, { 9.33f, -9.66f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 2.40f,  0.00f}
        };
        MKL_Complex8 b[LDB*N] = {
           {-5.69f, -8.21f}, {-2.83f,  6.46f},
           {-3.57f,  1.99f}, {-7.64f,  1.10f},
           { 8.42f, -9.83f}, {-2.33f, -4.23f},
           {-5.00f,  3.85f}, { 6.48f, -3.81f}
        };
        /* Executable statements */
        printf( "LAPACKE_chesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_chesv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_chesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_chesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_chesv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a Hermitian matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
   (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
   (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
   ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

   and B is the right-hand side matrix:

   ( -5.69, -8.21) ( -2.83,  6.46)
   ( -3.57,  1.99) ( -7.64,  1.10)
   (  8.42, -9.83) ( -2.33, -4.23)
   ( -5.00,  3.85) (  6.48, -3.81)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n Hermitian matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UH or
   A = L*D*LH, where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is Hermitian and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 LAPACKE_chesv (column-major, high-level) Example Program Results
```

```
 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex8 a[LDA*N] = {
           {-2.90f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           { 0.31f,  4.46f}, {-7.93f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           { 9.66f, -5.66f}, { 9.55f, -4.62f}, { 0.30f,  0.00f}, { 0.00f,  0.00f},
           {-2.28f,  2.14f}, {-3.51f,  3.11f}, { 9.33f, -9.66f}, { 2.40f,  0.00f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           {-5.69f, -8.21f}, {-3.57f,  1.99f}, { 8.42f, -9.83f}, {-5.00f,  3.85f},
           {-2.83f,  6.46f}, {-7.64f,  1.10f}, {-2.33f, -4.23f}, { 6.48f, -3.81f}
        };
        /* Executable statements */
        printf( "LAPACKE_chesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_chesv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
```

```
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_chesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

ZHESV Example

NOTE
For a detailed description and reference information on this function, please visit:

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves the complex system of linear equations for *X*:

```
A*X = B
```

where

- *A* is a square Hermitian matrix.
- The columns of matrix *B* are individual right-hand sides.
- The columns of *X* are the corresponding solutions.

The matrix *B* is overwritten by *X*.

The diagonal pivoting method is used to factor *A* as:

$A = U*D*U^H$ or $A = L*D*L^H$

where

- *U* (or *L*) is a product of permutation and unit upper (lower) triangular matrices.
- *D* is a Hermitian and block diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

The factored form of *A* is then used to solve the system of equations $A*X = B$.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major[1] Data Layout
- Example program in C for Column Major[1] Data Layout

See Also

?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - C
?hesv function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZHESV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZHESV Example.
*  ==============
*
*  The program computes the solution to the system of linear equations
*  with a Hermitian matrix A and multiple right-hand sides B,
*  where A is the coefficient matrix:
*
*  ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
*  (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
*  (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
*  ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)
*
*  and B is the right-hand side matrix:
*
*  ( -5.69, -8.21) ( -2.83,  6.46)
*  ( -3.57,  1.99) ( -7.64,  1.10)
*  (  8.42, -9.83) ( -2.33, -4.23)
*  ( -5.00,  3.85) (  6.48, -3.81)
*
*  Description.
*  ============
*
*  The routine solves for X the complex system of linear equations A*X = B,
*  where A is an n-by-n Hermitian matrix, the columns of matrix B are
*  individual right-hand sides, and the columns of X are the corresponding
*  solutions.
```

```
*
*  The diagonal pivoting method is used to factor A as A = U*D*UH or
*  A = L*D*LH, where U (or L) is a product of permutation and unit upper
*  (lower) triangular matrices, and D is Hermitian and block diagonal with
*  1-by-1 and 2-by-2 diagonal blocks.
*
*  The factored form of A is then used to solve the system of equations A*X = B.
*
*  Example Program Results.
*  ========================
*
* ZHESV Example Program Results
*
* Solution
* (  0.22, -0.95) ( -1.13,  0.18)
* ( -1.42, -1.30) (  0.70,  1.13)
* ( -0.65, -0.40) (  0.04,  0.07)
* ( -0.48,  1.35) (  1.15, -0.27)
*
* Details of factorization
* (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
* (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
* (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
* (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)
*
* Pivot indices
*     -1     -1     -3     -3
*  ==============================================================================
*
*     .. Parameters ..
      INTEGER          N, NRHS
      PARAMETER        ( N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = N, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      INTEGER          IPIV( N )
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ (-2.90, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ ( 0.31, 4.46),(-7.93, 0.00),( 0.00, 0.00),( 0.00, 0.00),
     $ ( 9.66,-5.66),( 9.55,-4.62),( 0.30, 0.00),( 0.00, 0.00),
     $ (-2.28, 2.14),(-3.51, 3.11),( 9.33,-9.66),( 2.40, 0.00)
     $                 /
      DATA             B/
     $ (-5.69,-8.21),(-3.57, 1.99),( 8.42,-9.83),(-5.00, 3.85),
     $ (-2.83, 6.46),(-7.64, 1.10),(-2.33,-4.23),( 6.48,-3.81)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZHESV
      EXTERNAL         PRINT_MATRIX, PRINT_INT_VECTOR
*
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZHESV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZHESV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL ZHESV( 'Upper', N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK,
     $            INFO )
*
*     Check for the exact singularity.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The element of the diagonal factor '
         WRITE(*,*)'D(',INFO,',',INFO,') is zero, so that'
         WRITE(*,*)'D is singular; the solution could not be computed.'
         STOP
      END IF
*
*     Print solution.
*
      CALL PRINT_MATRIX( 'Solution', N, NRHS, B, LDB )
*
*     Print details of factorization.
*
      CALL PRINT_MATRIX( 'Details of factorization', N, N, A, LDA )
*
*     Print pivot indices.
*
      CALL PRINT_INT_VECTOR( 'Pivot indices', N, IPIV )
      STOP
      END
*
*     End of ZHESV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
```

```
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a vector of integers.
*
      SUBROUTINE PRINT_INT_VECTOR( DESC, N, A )
      CHARACTER*(*)    DESC
      INTEGER          N
      INTEGER          A( N )
*
      INTEGER          I
*
      WRITE(*,*)
      WRITE(*,*) DESC
      WRITE(*,9999) ( A( I ), I = 1, N )
*
 9999 FORMAT( 11(:,1X,I6) )
      RETURN
      END
```

ZHESV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZHESV Example.
   ==============

   The program computes the solution to the system of linear equations
   with a Hermitian matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
   (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
   (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
   ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

   and B is the right-hand side matrix:

   ( -5.69, -8.21) ( -2.83,  6.46)
```

```
   ( -3.57,  1.99) ( -7.64,  1.10)
   (  8.42, -9.83) ( -2.33, -4.23)
   ( -5.00,  3.85) (  6.48, -3.81)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n Hermitian matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UH or
   A = L*D*LH, where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is Hermitian and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   The factored form of A is then used to solve the system of equations A*X = B.

   Example Program Results.
   ========================

 ZHESV Example Program Results

 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZHESV prototype */
extern void zhesv( char* uplo, int* n, int* nrhs, dcomplex* a, int* lda,
                int* ipiv, dcomplex* b, int* ldb, dcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_int_vector( char* desc, int n, int* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N
```

```
/* Main program */
int main() {
        /* Locals */
        int n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        int ipiv[N];
        dcomplex a[LDA*N] = {
           {-2.90,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           { 0.31,  4.46}, {-7.93,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           { 9.66, -5.66}, { 9.55, -4.62}, { 0.30,  0.00}, { 0.00,  0.00},
           {-2.28,  2.14}, {-3.51,  3.11}, { 9.33, -9.66}, { 2.40,  0.00}
        };
        dcomplex b[LDB*NRHS] = {
           {-5.69, -8.21}, {-3.57,  1.99}, { 8.42, -9.83}, {-5.00,  3.85},
           {-2.83,  6.46}, {-7.64,  1.10}, {-2.33, -4.23}, { 6.48, -3.81}
        };
        /* Executable statements */
        printf( " ZHESV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zhesv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, &wkopt, &lwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve the equations A*X = B */
        zhesv( "Upper", &n, &nrhs, a, &lda, ipiv, b, &ldb, work, &lwork, &info );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZHESV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
```

```
void print_int_vector( char* desc, int n, int* a ) {
        int j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zhesv Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zhesv Example.
   ======================

   The program computes the solution to the system of linear equations
   with a Hermitian matrix A and multiple right-hand sides B,
   where A is the coefficient matrix:

   ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
   (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
   (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
   ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

   and B is the right-hand side matrix:

   ( -5.69, -8.21) ( -2.83,  6.46)
   ( -3.57,  1.99) ( -7.64,  1.10)
   (  8.42, -9.83) ( -2.33, -4.23)
   ( -5.00,  3.85) (  6.48, -3.81)

   Description.
   ============

   The routine solves for X the complex system of linear equations A*X = B,
   where A is an n-by-n Hermitian matrix, the columns of matrix B are
   individual right-hand sides, and the columns of X are the corresponding
   solutions.

   The diagonal pivoting method is used to factor A as A = U*D*UH or
   A = L*D*LH, where U (or L) is a product of permutation and unit upper
   (lower) triangular matrices, and D is Hermitian and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.
```

```
  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_zhesv (row-major, high-level) Example Program Results

 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           {-2.90,  0.00}, { 0.31,  4.46}, { 9.66, -5.66}, {-2.28,  2.14},
           { 0.00,  0.00}, {-7.93,  0.00}, { 9.55, -4.62}, {-3.51,  3.11},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.30,  0.00}, { 9.33, -9.66},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 2.40,  0.00}
        };
        MKL_Complex16 b[LDB*N] = {
           {-5.69, -8.21}, {-2.83,  6.46},
           {-3.57,  1.99}, {-7.64,  1.10},
           { 8.42, -9.83}, {-2.33, -4.23},
           {-5.00,  3.85}, { 6.48, -3.81}
        };
        /* Executable statements */
        printf( "LAPACKE_zhesv (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
```

```
        info = LAPACKE_zhesv( LAPACK_ROW_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zhesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

LAPACKE_zhesv Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zhesv Example.
```

```
  ======================

  The program computes the solution to the system of linear equations
  with a Hermitian matrix A and multiple right-hand sides B,
  where A is the coefficient matrix:

  ( -2.90,  0.00) (  0.31,  4.46) (  9.66, -5.66) ( -2.28,  2.14)
  (  0.31, -4.46) ( -7.93,  0.00) (  9.55, -4.62) ( -3.51,  3.11)
  (  9.66,  5.66) (  9.55,  4.62) (  0.30,  0.00) (  9.33, -9.66)
  ( -2.28, -2.14) ( -3.51, -3.11) (  9.33,  9.66) (  2.40,  0.00)

  and B is the right-hand side matrix:

  ( -5.69, -8.21) ( -2.83,  6.46)
  ( -3.57,  1.99) ( -7.64,  1.10)
  (  8.42, -9.83) ( -2.33, -4.23)
  ( -5.00,  3.85) (  6.48, -3.81)

  Description.
  ============

  The routine solves for X the complex system of linear equations A*X = B,
  where A is an n-by-n Hermitian matrix, the columns of matrix B are
  individual right-hand sides, and the columns of X are the corresponding
  solutions.

  The diagonal pivoting method is used to factor A as A = U*D*UH or
  A = L*D*LH, where U (or L) is a product of permutation and unit upper
  (lower) triangular matrices, and D is Hermitian and block diagonal with
  1-by-1 and 2-by-2 diagonal blocks.

  The factored form of A is then used to solve the system of equations A*X = B.

  Example Program Results.
  ========================

 LAPACKE_zhesv (column-major, high-level) Example Program Results

 Solution
 (  0.22, -0.95) ( -1.13,  0.18)
 ( -1.42, -1.30) (  0.70,  1.13)
 ( -0.65, -0.40) (  0.04,  0.07)
 ( -0.48,  1.35) (  1.15, -0.27)

 Details of factorization
 (  3.17,  0.00) (  7.32,  3.28) ( -0.36,  0.06) (  0.20, -0.82)
 (  0.00,  0.00) (  0.03,  0.00) ( -0.48,  0.03) (  0.25, -0.76)
 (  0.00,  0.00) (  0.00,  0.00) (  0.30,  0.00) (  9.33, -9.66)
 (  0.00,  0.00) (  0.00,  0.00) (  0.00,  0.00) (  2.40,  0.00)

 Pivot indices
     -1     -1     -3     -3
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
```

```
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_int_vector( char* desc, MKL_INT n, MKL_INT* a );

/* Parameters */
#define N 4
#define NRHS 2
#define LDA N
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_INT ipiv[N];
        MKL_Complex16 a[LDA*N] = {
           {-2.90,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           { 0.31,  4.46}, {-7.93,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           { 9.66, -5.66}, { 9.55, -4.62}, { 0.30,  0.00}, { 0.00,  0.00},
           {-2.28,  2.14}, {-3.51,  3.11}, { 9.33, -9.66}, { 2.40,  0.00}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           {-5.69, -8.21}, {-3.57,  1.99}, { 8.42, -9.83}, {-5.00,  3.85},
           {-2.83,  6.46}, {-7.64,  1.10}, {-2.33, -4.23}, { 6.48, -3.81}
        };
        /* Executable statements */
        printf( "LAPACKE_zhesv (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zhesv( LAPACK_COL_MAJOR, 'U', n, nrhs, a, lda, ipiv,
                        b, ldb );
        /* Check for the exact singularity */
        if( info > 0 ) {
                printf( "The element of the diagonal factor " );
                printf( "D(%i,%i) is zero, so that D is singular;\n", info, info );
                printf( "the solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print solution */
        print_matrix( "Solution", n, nrhs, b, ldb );
        /* Print details of factorization */
        print_matrix( "Details of factorization", n, n, a, lda );
        /* Print pivot indices */
        print_int_vector( "Pivot indices", n, ipiv );
        exit( 0 );
} /* End of LAPACKE_zhesv Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a vector of integers */
```

```
void print_int_vector( char* desc, MKL_INT n, MKL_INT* a ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) printf( " %6i", a[j] );
        printf( "\n" );
}
```

Least Squares and Eigenvalue Problems

The Intel® oneAPI Math Kernel Library includes LAPACK routines to solve least-squares problems, eigenvalue and singular value problems, and Sylvester equations.

To solve a particular problem, you can call two or more **computational routines** or call a corresponding **driver routine** that combines several tasks in one call, such as `?gesv` for factoring and solving. For example, to solve a system of linear equations with a general matrix, call `?getrf` (*LU* factorization) and then `?getrs` (computing the solution). Call `?gerfs` to refine the solution and get the error bounds. Alternatively, use the driver routine `?gesvx`, which performs all these tasks in one call.

You can also find an appropriate routine using the **characteristics** of your data and the **operations** you need by using the oneMKL LAPACK Function Finding Advisor, which helps you find the routine that suits your needs best.

Caution
The standard LAPACK functions do not check the input data (matrices) for IEEE 754 floating point INFs or NANS. INFs and NANs will propagate through the computations and may cause unexpected results or instabilities. It is the user's responsibility to ensure that the input data do not contain INFs nor NaNs.

The LAPACKE (C interfaces to LAPACK) functions do check for NaNs in the input data (matrices) , and an error code is returned if a NAN is found.

Driver Routines

- Linear Least Squares (LLS) Problems
- Symmetric Eigenproblems
- Nonsymmetric Eigenproblems
- Singular Value Decomposition

Linear Least Squares (LLS) Problems

?gels	Uses *QR* or *LQ* factorization to solve an overdetermined or underdetermined linear system with a full rank matrix.	cgels dgels sgels zgels
?gelsd	Computes the minimum norm solution to a linear least squares problem using the singular value decomposition of *A* and a divide and conquer method.	cgelsd dgelsd sgelsd zgelsd

?gels function

Uses *QR* or *LQ* factorization to solve an overdetermined or underdetermined linear system with full rank matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call dgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call cgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call zgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

Calling from C:

```
sgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info);
dgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info);
cgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info);
zgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info);
```

C Interface

```
lapack_int LAPACKE_sgels(int matrix_layout, char trans, lapack_int m, lapack_int n,
lapack_int nrhs, float* a, lapack_int lda, float* b, lapack_int ldb);
lapack_int LAPACKE_dgels(int matrix_layout, char trans, lapack_int m, lapack_int n,
lapack_int nrhs, double* a, lapack_int lda, double* b, lapack_int ldb);
lapack_int LAPACKE_cgels(int matrix_layout, char trans, lapack_int m, lapack_int n,
lapack_int nrhs, lapack_complex_float* a, lapack_int lda, lapack_complex_float* b,
lapack_int ldb);
lapack_int LAPACKE_zgels(int matrix_layout, char trans, lapack_int m, lapack_int n,
lapack_int nrhs, lapack_complex_double* a, lapack_int lda, lapack_complex_double* b,
lapack_int ldb);
```

See Also

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Notational Conventions

C Interface Conventions

CGELS Example

NOTE
For a detailed description and reference information on this function, please visit:

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves overdetermined or underdetermined complex linear systems involving a rectangular matrix *A*, or its transpose, using a *QR* or *LQ* factorization of *A*. The routine assumes that *A* has full rank.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGELS Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CGELS Example.
*  ==============
*
*  Program computes the minimum norm solution to the underdetermined linear
*  system A*X = B with full rank matrix A using LQ factorization,
*  where A is the coefficient matrix:
*
*  ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
*  ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
*  ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)
*
*  and B is the right-hand side matrix:
*
*  ( -7.02,  4.80) (  3.88, -2.59)
*  (  0.62, -2.40) (  1.57,  3.24)
*  (  3.10, -2.19) ( -6.93, -5.99)
*
*  Description.
*  ============
*
*  The routine solves overdetermined or underdetermined complex linear systems
*  involving an m-by-n matrix A, or its transpose, using a QR or LQ
*  factorization of A. It is assumed that A has full rank.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
```

```
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  Example Program Results.
*  ========================
*
* CGELS Example Program Results
*
* Minimum norm solution
* ( -0.25, -0.04) ( -0.21,  0.42)
* (  0.99,  0.27) ( -0.21, -0.43)
* (  0.25,  0.43) ( -0.24, -0.13)
* ( -0.32,  0.14) ( -0.23, -0.09)
*
* Details of LQ factorization
* ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
* (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
* (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 3, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      COMPLEX          A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ (-4.20,-3.44),(-5.43,-8.81),(-5.56, 3.39),
     $ (-3.35, 1.52),(-4.53,-8.47),( 2.90,-9.22),
     $ ( 1.73, 8.85),( 5.93, 3.75),( 8.03, 9.37),
     $ ( 2.35, 0.34),(-3.75,-5.66),( 5.69,-0.47)
     $                 /
      DATA             B/
     $ (-7.02, 4.80),( 0.62,-2.40),( 3.10,-2.19),( 0.00, 0.00),
     $ ( 3.88,-2.59),( 1.57, 3.24),(-6.93,-5.99),( 0.00, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CGELS
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CGELS Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
```

```
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL CGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
*
*     Check for the full rank.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element ',INFO,' of the triangular '
         WRITE(*,*)'factor of A is zero, so that A does not have full '
         WRITE(*,*)'rank; the minimum norm solution could not be '
         WRITE(*,*)'computed.'
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print details of LQ factorization.
*
      CALL PRINT_MATRIX( 'Details of LQ factorization', M, N, A, LDA )
      STOP
      END
*
*     End of CGELS Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

CGELS Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
```

```
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   CGELS Example.
   ==============

   Program computes the minimum norm solution to the underdetermined linear
   system A*X = B with full rank matrix A using LQ factorization,
   where A is the coefficient matrix:

   ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
   ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
   ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

   and B is the right-hand side matrix:

   ( -7.02,  4.80) (  3.88, -2.59)
   (  0.62, -2.40) (  1.57,  3.24)
   (  3.10, -2.19) ( -6.93, -5.99)

   Description.
   ============

   The routine solves overdetermined or underdetermined complex linear systems
   involving an m-by-n matrix A, or its transpose, using a QR or LQ
   factorization of A. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 CGELS Example Program Results

 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)

 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>
```

```
/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGELS prototype */
extern void cgels( char* trans, int* m, int* n, int* nrhs, fcomplex* a,
                int* lda, fcomplex* b, int* ldb, fcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        fcomplex a[LDA*N] = {
           {-4.20f, -3.44f}, {-5.43f, -8.81f}, {-5.56f,  3.39f},
           {-3.35f,  1.52f}, {-4.53f, -8.47f}, { 2.90f, -9.22f},
           { 1.73f,  8.85f}, { 5.93f,  3.75f}, { 8.03f,  9.37f},
           { 2.35f,  0.34f}, {-3.75f, -5.66f}, { 5.69f, -0.47f}
        };
        fcomplex b[LDB*NRHS] = {
           {-7.02f,  4.80f}, { 0.62f, -2.40f}, { 3.10f, -2.19f}, { 0.00f,  0.00f},
           { 3.88f, -2.59f}, { 1.57f,  3.24f}, {-6.93f, -5.99f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
        printf( " CGELS Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, &wkopt, &lwork,
                        &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve the equations A*X = B */
        cgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, work, &lwork,
                        &info );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        /* Free workspace */
```

```
        free( (void*)work );
        exit( 0 );
} /* End of CGELS Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_cgels Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgels Example.
   ======================

   Program computes the minimum norm solution to the underdetermined linear
   system A*X = B with full rank matrix A using LQ factorization,
   where A is the coefficient matrix:

   ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
   ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
   ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

   and B is the right-hand side matrix:

   ( -7.02,  4.80) (  3.88, -2.59)
   (  0.62, -2.40) (  1.57,  3.24)
   (  3.10, -2.19) ( -6.93, -5.99)

   Description.
   ============

   The routine solves overdetermined or underdetermined complex linear systems
   involving an m-by-n matrix A, or its transpose, using a QR or LQ
   factorization of A. It is assumed that A has full rank.
```

```
  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  Example Program Results.
  ========================

 LAPACKE_cgels (row-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)

 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex8 a[LDA*M] = {
           {-4.20f, -3.44f}, {-3.35f,  1.52f}, { 1.73f,  8.85f}, { 2.35f,  0.34f},
           {-5.43f, -8.81f}, {-4.53f, -8.47f}, { 5.93f,  3.75f}, {-3.75f, -5.66f},
           {-5.56f,  3.39f}, { 2.90f, -9.22f}, { 8.03f,  9.37f}, { 5.69f, -0.47f}
        };
        MKL_Complex8 b[LDB*N] = {
           {-7.02f,  4.80f}, { 3.88f, -2.59f},
           { 0.62f, -2.40f}, { 1.57f,  3.24f},
           { 3.10f, -2.19f}, {-6.93f, -5.99f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgels (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgels( LAPACK_ROW_MAJOR, 'N', m, n, nrhs, a, lda, b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
```

```
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

LAPACKE_cgels Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgels Example.
   ======================

   Program computes the minimum norm solution to the underdetermined linear
   system A*X = B with full rank matrix A using LQ factorization,
   where A is the coefficient matrix:

   ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
   ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
   ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

   and B is the right-hand side matrix:

   ( -7.02,  4.80) (  3.88, -2.59)
   (  0.62, -2.40) (  1.57,  3.24)
   (  3.10, -2.19) ( -6.93, -5.99)
```

```
  Description.
  ============

  The routine solves overdetermined or underdetermined complex linear systems
  involving an m-by-n matrix A, or its transpose, using a QR or LQ
  factorization of A. It is assumed that A has full rank.

  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  Example Program Results.
  ========================

 LAPACKE_cgels (column-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)

 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex8 a[LDA*N] = {
           {-4.20f, -3.44f}, {-5.43f, -8.81f}, {-5.56f,  3.39f},
           {-3.35f,  1.52f}, {-4.53f, -8.47f}, { 2.90f, -9.22f},
           { 1.73f,  8.85f}, { 5.93f,  3.75f}, { 8.03f,  9.37f},
           { 2.35f,  0.34f}, {-3.75f, -5.66f}, { 5.69f, -0.47f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           {-7.02f,  4.80f}, { 0.62f, -2.40f}, { 3.10f, -2.19f}, { 0.00f,  0.00f},
           { 3.88f, -2.59f}, { 1.57f,  3.24f}, {-6.93f, -5.99f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
```

```
        printf( "LAPACKE_cgels (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgels( LAPACK_COL_MAJOR, 'N', m, n, nrhs, a, lda, b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

DGELS Example

NOTE
For a detailed description and reference information on this function, please visit:

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves overdetermined or underdetermined real linear systems involving a rectangular matrix *A*, or its transpose, using a *QR* or *LQ* factorization of *A*. The routine assumes that *A* has full rank.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGELS Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DGELS Example.
*  ==============
*
*  Program computes the least squares solution to the overdetermined linear
*  system A*X = B with full rank matrix A using QR factorization,
*  where A is the coefficient matrix:
*
*    1.44  -7.84  -4.39   4.53
*   -9.96  -0.28  -3.24   3.83
*   -7.55   3.24   6.27  -6.64
*    8.34   8.09   5.28   2.06
*    7.08   2.52   0.74  -2.47
*   -5.45  -5.70  -1.19   4.70
*
*  and B is the right-hand side matrix:
*
*    8.58   9.35
*    8.26  -4.43
*    8.48  -0.70
*   -5.28  -0.26
*    5.72  -7.36
*    8.93  -2.52
*
*  Description.
*  ============
*
*  The routine 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. It is assumed that A has full rank.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  Example Program Results.
*  ========================
*
* DGELS Example Program Results
*
* Solution
```

```
*  -0.45   0.25
*  -0.85  -0.90
*   0.71   0.63
*   0.13   0.14
*
* Residual sum of squares for the solution
* 195.36 107.06
*
* Details of QR factorization
* -17.54  -4.76  -1.96   0.42
*  -0.52  12.40   7.88  -5.84
*  -0.40  -0.14  -5.75   4.11
*   0.44  -0.66  -0.20  -7.78
*   0.37  -0.26  -0.17  -0.15
*  -0.29   0.46   0.41   0.24
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 6, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = M )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $  1.44,-9.96,-7.55, 8.34, 7.08,-5.45,
     $ -7.84,-0.28, 3.24, 8.09, 2.52,-5.70,
     $ -4.39,-3.24, 6.27, 5.28, 0.74,-1.19,
     $  4.53, 3.83,-6.64, 2.06,-2.47, 4.70
     $                 /
      DATA             B/
     $  8.58, 8.26, 8.48,-5.28, 5.72, 8.93,
     $  9.35,-4.43,-0.70,-0.26,-7.36,-2.52
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         DGELS
      EXTERNAL         PRINT_MATRIX, PRINT_VECTOR_NORM
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DGELS Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
```

```
*     Solve the equations A*X = B.
*
      CALL DGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
*
*     Check for the full rank.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element ',INFO,' of the triangular '
         WRITE(*,*)'factor of A is zero, so that A does not have full '
         WRITE(*,*)'rank; the least squares solution could not be '
         WRITE(*,*)'computed.'
         STOP
      END IF
*
*     Print least squares solution.
*
      CALL PRINT_MATRIX( 'Least squares solution', N, NRHS, B, LDB )
*
*     Print residual sum of squares for the solution
*
      CALL PRINT_VECTOR_NORM(
     $     'Residual sum of squares for the solution', M-N, NRHS,
     $     B( N+1, 1 ), LDB )
*
*     Print details of QR factorization.
*
      CALL PRINT_MATRIX( 'Details of QR factorization', M, N, A, LDA )
      STOP
      END
*
*     End of DGELS Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing norms of matrix columns.
*
      SUBROUTINE PRINT_VECTOR_NORM( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
```

```
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      DOUBLE PRECISION TEMP
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO J = 1, N
         TEMP = 0.0
         DO I = 1, M
            TEMP = TEMP + A( I, J )*A( I, J )
         END DO
         WRITE(*,9998,ADVANCE='NO') TEMP
      END DO
      WRITE(*,*)
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DGELS Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DGELS Example.
   ==============

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
```

```
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 DGELS Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14

 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>

/* DGELS prototype */
extern void dgels( char* trans, int* m, int* n, int* nrhs, double* a, int* lda,
                double* b, int* ldb, double* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );
extern void print_vector_norm( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA M
#define LDB M

/* Main program */
int main() {
```

```
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        double wkopt;
        double* work;
        /* Local arrays */
        double a[LDA*N] = {
            1.44, -9.96, -7.55,  8.34,  7.08, -5.45,
           -7.84, -0.28,  3.24,  8.09,  2.52, -5.70,
           -4.39, -3.24,  6.27,  5.28,  0.74, -1.19,
            4.53,  3.83, -6.64,  2.06, -2.47,  4.70
        };
        double b[LDB*NRHS] = {
            8.58,  8.26,  8.48, -5.28,  5.72,  8.93,
            9.35, -4.43, -0.70, -0.26, -7.36, -2.52
        };
        /* Executable statements */
        printf( " DGELS Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, &wkopt, &lwork,
                        &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve the equations A*X = B */
        dgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, work, &lwork,
                        &info );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n], ldb );
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DGELS Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
```

```
        double norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i+j*lda] * a[i+j*lda];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

LAPACKE_dgels Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgels Example.
   ======================

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_dgels (row-major, high-level) Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14

 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        double a[LDA*M] = {
            1.44, -7.84, -4.39,  4.53,
           -9.96, -0.28, -3.24,  3.83,
           -7.55, 3.24, 6.27, -6.64,
            8.34, 8.09, 5.28,  2.06,
            7.08, 2.52, 0.74, -2.47,
           -5.45, -5.70, -1.19,  4.70
        };
        double b[LDB*M] = {
            8.58, 9.35,
```

```
            8.26, -4.43,
            8.48, -0.70,
           -5.28, -0.26,
            5.72, -7.36,
            8.93, -2.52
        };
        /* Executable statements */
        printf( "LAPACKE_dgels (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgels( LAPACK_ROW_MAJOR, 'N', m, n, nrhs, a, lda,
                        b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n*ldb], ldb );
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        double norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i*lda+j] * a[i*lda+j];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

LAPACKE_dgels Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
```

```
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgels Example.
   ======================

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_dgels (column-major, high-level) Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14
```

```
 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );
extern void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA M
#define LDB M

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        double a[LDA*N] = {
            1.44, -9.96, -7.55,  8.34,  7.08, -5.45,
           -7.84, -0.28,  3.24,  8.09,  2.52, -5.70,
           -4.39, -3.24,  6.27,  5.28,  0.74, -1.19,
            4.53,  3.83, -6.64,  2.06, -2.47,  4.70
        };
        double b[LDB*NRHS] = {
            8.58,  8.26,  8.48, -5.28,  5.72,  8.93,
            9.35, -4.43, -0.70, -0.26, -7.36, -2.52
        };
        /* Executable statements */
        printf( "LAPACKE_dgels (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgels( LAPACK_COL_MAJOR, 'N', m, n, nrhs, a, lda,
                        b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n], ldb );
```

```
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        double norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i+j*lda] * a[i+j*lda];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

SGELS Example

NOTE
For a detailed description and reference information on this function, please visit:

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves overdetermined or underdetermined real linear systems involving a rectangular matrix *A*, or its transpose, using a *QR* or *LQ* factorization of *A*. The routine assumes that *A* has full rank.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGELS Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SGELS Example.
*  ==============
*
*  Program computes the least squares solution to the overdetermined linear
*  system A*X = B with full rank matrix A using QR factorization,
*  where A is the coefficient matrix:
*
*    1.44  -7.84  -4.39   4.53
*   -9.96  -0.28  -3.24   3.83
*   -7.55   3.24   6.27  -6.64
*    8.34   8.09   5.28   2.06
*    7.08   2.52   0.74  -2.47
*   -5.45  -5.70  -1.19   4.70
*
*  and B is the right-hand side matrix:
*
*    8.58   9.35
*    8.26  -4.43
*    8.48  -0.70
*   -5.28  -0.26
*    5.72  -7.36
*    8.93  -2.52
*
*  Description.
*  ============
*
*  The routine 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. It is assumed that A has full rank.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  Example Program Results.
*  ========================
*
* SGELS Example Program Results
*
* Solution
```

```
*  -0.45   0.25
*  -0.85  -0.90
*   0.71   0.63
*   0.13   0.14
*
* Residual sum of squares for the solution
* 195.36 107.06
*
* Details of QR factorization
* -17.54  -4.76  -1.96   0.42
*  -0.52  12.40   7.88  -5.84
*  -0.40  -0.14  -5.75   4.11
*   0.44  -0.66  -0.20  -7.78
*   0.37  -0.26  -0.17  -0.15
*  -0.29   0.46   0.41   0.24
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 6, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = M )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      REAL             A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $  1.44,-9.96,-7.55, 8.34, 7.08,-5.45,
     $ -7.84,-0.28, 3.24, 8.09, 2.52,-5.70,
     $ -4.39,-3.24, 6.27, 5.28, 0.74,-1.19,
     $  4.53, 3.83,-6.64, 2.06,-2.47, 4.70
     $                  /
      DATA             B/
     $  8.58, 8.26, 8.48,-5.28, 5.72, 8.93,
     $  9.35,-4.43,-0.70,-0.26,-7.36,-2.52
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SGELS
      EXTERNAL         PRINT_MATRIX, PRINT_VECTOR_NORM
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SGELS Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
```

```
*     Solve the equations A*X = B.
*
      CALL SGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
*
*     Check for the full rank.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element ',INFO,' of the triangular '
         WRITE(*,*)'factor of A is zero, so that A does not have full '
         WRITE(*,*)'rank; the least squares solution could not be '
         WRITE(*,*)'computed.'
         STOP
      END IF
*
*     Print least squares solution.
*
      CALL PRINT_MATRIX( 'Least squares solution', N, NRHS, B, LDB )
*
*     Print residual sum of squares for the solution
*
      CALL PRINT_VECTOR_NORM(
     $     'Residual sum of squares for the solution', M-N, NRHS,
     $     B( N+1, 1 ), LDB )
*
*     Print details of QR factorization.
*
      CALL PRINT_MATRIX( 'Details of QR factorization', M, N, A, LDA )
      STOP
      END
*
*     End of SGELS Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
*
*     Auxiliary routine: printing norms of matrix columns.
*
      SUBROUTINE PRINT_VECTOR_NORM( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
```

```
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      REAL             TEMP
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO J = 1, N
         TEMP = 0.0
         DO I = 1, M
            TEMP = TEMP + A( I, J )*A( I, J )
         END DO
         WRITE(*,9998,ADVANCE='NO') TEMP
      END DO
      WRITE(*,*)
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SGELS Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SGELS Example.
   ==============

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
```

```
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 SGELS Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14

 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>

/* SGELS prototype */
extern void sgels( char* trans, int* m, int* n, int* nrhs, float* a, int* lda,
                float* b, int* ldb, float* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );
extern void print_vector_norm( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA M
#define LDB M

/* Main program */
int main() {
```

```
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
        float a[LDA*N] = {
            1.44f, -9.96f, -7.55f,  8.34f,  7.08f, -5.45f,
           -7.84f, -0.28f,  3.24f,  8.09f,  2.52f, -5.70f,
           -4.39f, -3.24f,  6.27f,  5.28f,  0.74f, -1.19f,
            4.53f,  3.83f, -6.64f,  2.06f, -2.47f,  4.70f
        };
        float b[LDB*NRHS] = {
            8.58f,  8.26f,  8.48f, -5.28f,  5.72f,  8.93f,
            9.35f, -4.43f, -0.70f, -0.26f, -7.36f, -2.52f
        };
        /* Executable statements */
        printf( " SGELS Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        sgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, &wkopt, &lwork,
                        &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve the equations A*X = B */
        sgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, work, &lwork,
                        &info );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n], ldb );
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SGELS Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
```

```
        float norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i+j*lda] * a[i+j*lda];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

LAPACKE_sgels Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgels Example.
   ======================

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_sgels (row-major, high-level) Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14

 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        float a[LDA*M] = {
            1.44f, -7.84f, -4.39f,  4.53f,
           -9.96f, -0.28f, -3.24f,  3.83f,
           -7.55f, 3.24f, 6.27f, -6.64f,
            8.34f, 8.09f, 5.28f,  2.06f,
            7.08f, 2.52f, 0.74f, -2.47f,
           -5.45f, -5.70f, -1.19f,  4.70f
        };
        float b[LDB*M] = {
            8.58f, 9.35f,
```

```
            8.26f, -4.43f,
            8.48f, -0.70f,
           -5.28f, -0.26f,
            5.72f, -7.36f,
            8.93f, -2.52f
        };
        /* Executable statements */
        printf( "LAPACKE_sgels (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgels( LAPACK_ROW_MAJOR, 'N', m, n, nrhs, a, lda,
                        b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n*ldb], ldb );
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_sgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        float norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i*lda+j] * a[i*lda+j];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

LAPACKE_sgels Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
```

```
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   LAPACKE_sgels Example.
   ======================

   Program computes the least squares solution to the overdetermined linear
   system A*X = B with full rank matrix A using QR factorization,
   where A is the coefficient matrix:

     1.44  -7.84  -4.39   4.53
    -9.96  -0.28  -3.24   3.83
    -7.55   3.24   6.27  -6.64
     8.34   8.09   5.28   2.06
     7.08   2.52   0.74  -2.47
    -5.45  -5.70  -1.19   4.70

   and B is the right-hand side matrix:

     8.58   9.35
     8.26  -4.43
     8.48  -0.70
    -5.28  -0.26
     5.72  -7.36
     8.93  -2.52

   Description.
   ============

   The routine 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. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_sgels (column-major, high-level) Example Program Results

 Solution
  -0.45   0.25
  -0.85  -0.90
   0.71   0.63
   0.13   0.14
```

```
 Residual sum of squares for the solution
 195.36 107.06

 Details of QR factorization
 -17.54  -4.76  -1.96   0.42
  -0.52  12.40   7.88  -5.84
  -0.40  -0.14  -5.75   4.11
   0.44  -0.66  -0.20  -7.78
   0.37  -0.26  -0.17  -0.15
  -0.29   0.46   0.41   0.24
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );
extern void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define NRHS 2
#define LDA M
#define LDB M

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        float a[LDA*N] = {
            1.44f, -9.96f, -7.55f,  8.34f,  7.08f, -5.45f,
           -7.84f, -0.28f,  3.24f,  8.09f,  2.52f, -5.70f,
           -4.39f, -3.24f,  6.27f,  5.28f,  0.74f, -1.19f,
            4.53f,  3.83f, -6.64f,  2.06f, -2.47f,  4.70f
        };
        float b[LDB*NRHS] = {
            8.58f,  8.26f,  8.48f, -5.28f,  5.72f,  8.93f,
            9.35f, -4.43f, -0.70f, -0.26f, -7.36f, -2.52f
        };
        /* Executable statements */
        printf( "LAPACKE_sgels (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgels( LAPACK_COL_MAJOR, 'N', m, n, nrhs, a, lda,
                        b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print least squares solution */
        print_matrix( "Least squares solution", n, nrhs, b, ldb );
        /* Print residual sum of squares for the solution */
        print_vector_norm( "Residual sum of squares for the solution", m-n, nrhs,
                        &b[n], ldb );
```

```
        /* Print details of QR factorization */
        print_matrix( "Details of QR factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_sgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing norms of matrix columns */
void print_vector_norm( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        float norm;
        printf( "\n %s\n", desc );
        for( j = 0; j < n; j++ ) {
                norm = 0.0;
                for( i = 0; i < m; i++ ) norm += a[i+j*lda] * a[i+j*lda];
                printf( " %6.2f", norm );
        }
        printf( "\n" );
}
```

ZGELS Example

NOTE
For a detailed description and reference information on this function, please visit:

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine solves overdetermined or underdetermined complex linear systems involving a rectangular matrix *A*, or its transpose, using a *QR* or *LQ* factorization of *A*. The routine assumes that *A* has full rank.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gels function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGELS Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZGELS Example.
*  ==============
*
*  Program computes the minimum norm solution to the underdetermined linear
*  system A*X = B with full rank matrix A using LQ factorization,
*  where A is the coefficient matrix:
*
*  ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
*  ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
*  ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)
*
*  and B is the right-hand side matrix:
*
*  ( -7.02,  4.80) (  3.88, -2.59)
*  (  0.62, -2.40) (  1.57,  3.24)
*  (  3.10, -2.19) ( -6.93, -5.99)
*
*  Description.
*  ============
*
*  The routine solves overdetermined or underdetermined complex linear systems
*  involving an m-by-n matrix A, or its transpose, using a QR or LQ
*  factorization of A. It is assumed that A has full rank.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  Example Program Results.
*  ========================
*
* ZGELS Example Program Results
*
* Minimum norm solution
* ( -0.25, -0.04) ( -0.21,  0.42)
* (  0.99,  0.27) ( -0.21, -0.43)
* (  0.25,  0.43) ( -0.24, -0.13)
* ( -0.32,  0.14) ( -0.23, -0.09)
*
* Details of LQ factorization
```

```
* ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
* (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
* (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 3, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 100 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ (-4.20,-3.44),(-5.43,-8.81),(-5.56, 3.39),
     $ (-3.35, 1.52),(-4.53,-8.47),( 2.90,-9.22),
     $ ( 1.73, 8.85),( 5.93, 3.75),( 8.03, 9.37),
     $ ( 2.35, 0.34),(-3.75,-5.66),( 5.69,-0.47)
     $                /
      DATA             B/
     $ (-7.02, 4.80),( 0.62,-2.40),( 3.10,-2.19),( 0.00, 0.00),
     $ ( 3.88,-2.59),( 1.57, 3.24),(-6.93,-5.99),( 0.00, 0.00)
     $                /
*
*     .. External Subroutines ..
      EXTERNAL         ZGELS
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZGELS Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL ZGELS( 'No transpose', M, N, NRHS, A, LDA, B, LDB, WORK,
     $            LWORK, INFO )
*
*     Check for the full rank.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The diagonal element ',INFO,' of the triangular '
         WRITE(*,*)'factor of A is zero, so that A does not have full '
         WRITE(*,*)'rank; the minimum norm solution could not be '
         WRITE(*,*)'computed.'
```

```
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print details of LQ factorization.
*
      CALL PRINT_MATRIX( 'Details of LQ factorization', M, N, A, LDA )
      STOP
      END
*
*     End of ZGELS Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

ZGELS Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZGELS Example.
   ==============
```

```
  Program computes the minimum norm solution to the underdetermined linear
  system A*X = B with full rank matrix A using LQ factorization,
  where A is the coefficient matrix:

  ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
  ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
  ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

  and B is the right-hand side matrix:

  ( -7.02,  4.80) (  3.88, -2.59)
  (  0.62, -2.40) (  1.57,  3.24)
  (  3.10, -2.19) ( -6.93, -5.99)

  Description.
  ============

  The routine solves overdetermined or underdetermined complex linear systems
  involving an m-by-n matrix A, or its transpose, using a QR or LQ
  factorization of A. It is assumed that A has full rank.

  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  Example Program Results.
  ========================

 ZGELS Example Program Results

 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)

 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGELS prototype */
extern void zgels( char* trans, int* m, int* n, int* nrhs, dcomplex* a,
                int* lda, dcomplex* b, int* ldb, dcomplex* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );

/* Parameters */
#define M 3
#define N 4
```

```
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        dcomplex a[LDA*N] = {
           {-4.20, -3.44}, {-5.43, -8.81}, {-5.56,  3.39},
           {-3.35,  1.52}, {-4.53, -8.47}, { 2.90, -9.22},
           { 1.73,  8.85}, { 5.93,  3.75}, { 8.03,  9.37},
           { 2.35,  0.34}, {-3.75, -5.66}, { 5.69, -0.47}
        };
        dcomplex b[LDB*NRHS] = {
           {-7.02,  4.80}, { 0.62, -2.40}, { 3.10, -2.19}, { 0.00,  0.00},
           { 3.88, -2.59}, { 1.57,  3.24}, {-6.93, -5.99}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( " ZGELS Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, &wkopt, &lwork,
                        &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve the equations A*X = B */
        zgels( "No transpose", &m, &n, &nrhs, a, &lda, b, &ldb, work, &lwork,
                        &info );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZGELS Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_zgels Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgels Example.
   ======================

   Program computes the minimum norm solution to the underdetermined linear
   system A*X = B with full rank matrix A using LQ factorization,
   where A is the coefficient matrix:

   ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
   ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
   ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

   and B is the right-hand side matrix:

   ( -7.02,  4.80) (  3.88, -2.59)
   (  0.62, -2.40) (  1.57,  3.24)
   (  3.10, -2.19) ( -6.93, -5.99)

   Description.
   ============

   The routine solves overdetermined or underdetermined complex linear systems
   involving an m-by-n matrix A, or its transpose, using a QR or LQ
   factorization of A. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_zgels (row-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)
```

```
 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex16 a[LDA*M] = {
           {-4.20, -3.44}, {-3.35,  1.52}, { 1.73,  8.85}, { 2.35,  0.34},
           {-5.43, -8.81}, {-4.53, -8.47}, { 5.93,  3.75}, {-3.75, -5.66},
           {-5.56,  3.39}, { 2.90, -9.22}, { 8.03,  9.37}, { 5.69, -0.47}
        };
        MKL_Complex16 b[LDB*N] = {
           {-7.02,  4.80}, { 3.88, -2.59},
           { 0.62, -2.40}, { 1.57,  3.24},
           { 3.10, -2.19}, {-6.93, -5.99},
           { 0.00,  0.00}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zgels (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgels( LAPACK_ROW_MAJOR, 'N', m, n, nrhs, a, lda, b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zgels Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

LAPACKE_zgels Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgels Example.
   ======================

   Program computes the minimum norm solution to the underdetermined linear
   system A*X = B with full rank matrix A using LQ factorization,
   where A is the coefficient matrix:

   ( -4.20, -3.44) ( -3.35,  1.52) (  1.73,  8.85) (  2.35,  0.34)
   ( -5.43, -8.81) ( -4.53, -8.47) (  5.93,  3.75) ( -3.75, -5.66)
   ( -5.56,  3.39) (  2.90, -9.22) (  8.03,  9.37) (  5.69, -0.47)

   and B is the right-hand side matrix:

   ( -7.02,  4.80) (  3.88, -2.59)
   (  0.62, -2.40) (  1.57,  3.24)
   (  3.10, -2.19) ( -6.93, -5.99)

   Description.
   ============

   The routine solves overdetermined or underdetermined complex linear systems
   involving an m-by-n matrix A, or its transpose, using a QR or LQ
   factorization of A. It is assumed that A has full rank.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   Example Program Results.
   ========================

 LAPACKE_zgels (column-major, high-level) Example Program Results
```

```
 Minimum norm solution
 ( -0.25, -0.04) ( -0.21,  0.42)
 (  0.99,  0.27) ( -0.21, -0.43)
 (  0.25,  0.43) ( -0.24, -0.13)
 ( -0.32,  0.14) ( -0.23, -0.09)

 Details of LQ factorization
 ( 11.40,  0.00) (  0.18, -0.14) ( -0.23, -0.52) ( -0.15,  0.01)
 (  7.73, -0.39) ( 15.32,  0.00) ( -0.22,  0.42) (  0.45,  0.17)
 (  8.60, -5.68) (  3.96,  6.46) ( 12.54,  0.00) ( -0.02, -0.47)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info;
        /* Local arrays */
        MKL_Complex16 a[LDA*N] = {
           {-4.20, -3.44}, {-5.43, -8.81}, {-5.56,  3.39},
           {-3.35,  1.52}, {-4.53, -8.47}, { 2.90, -9.22},
           { 1.73,  8.85}, { 5.93,  3.75}, { 8.03,  9.37},
           { 2.35,  0.34}, {-3.75, -5.66}, { 5.69, -0.47}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           {-7.02,  4.80}, { 0.62, -2.40}, { 3.10, -2.19}, { 0.00,  0.00},
           { 3.88, -2.59}, { 1.57,  3.24}, {-6.93, -5.99}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zgels (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgels( LAPACK_COL_MAJOR, 'N', m, n, nrhs, a, lda, b, ldb );
        /* Check for the full rank */
        if( info > 0 ) {
                printf( "The diagonal element %i of the triangular factor ", info );
                printf( "of A is zero, so that A does not have full rank;\n" );
                printf( "the minimum norm solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print details of LQ factorization */
        print_matrix( "Details of LQ factorization", m, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zgels Example */
```

```
/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

?gelsd function

Computes the minimum-norm solution to a linear least squares problem using the singular value decomposition of *A* and a divide and conquer method.

NOTE
For a detailed description and reference information on this function, please visit:

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, iwork, info)
call dgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, iwork, info)
call cgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, rwork, iwork, info)
call zgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, rwork, iwork, info)
```

Calling from C:

```
sgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, iwork, info);
dgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, iwork, info);
cgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, rwork, iwork, info);
zgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork, rwork, iwork, info);
```

C Interface

```
lapack_int LAPACKE_sgelsd(int matrix_layout, lapack_int m, lapack_int n, lapack_int nrhs,
float* a, lapack_int lda, float* b, lapack_int ldb, float* s, float rcond, lapack_int*
rank);
lapack_int LAPACKE_dgelsd(int matrix_layout, lapack_int m, lapack_int n, lapack_int nrhs,
double* a, lapack_int lda, double* b, lapack_int ldb, double* s, double rcond, lapack_int*
rank);
lapack_int LAPACKE_cgelsd(int matrix_layout, lapack_int m, lapack_int n, lapack_int nrhs,
lapack_complex_float* a, lapack_int lda, lapack_complex_float* b, lapack_int ldb, float* s,
float rcond, lapack_int* rank);
lapack_int LAPACKE_zgelsd(int matrix_layout, lapack_int m, lapack_int n, lapack_int nrhs,
lapack_complex_double* a, lapack_int lda, lapack_complex_double* b, lapack_int ldb, double*
s, double rcond, lapack_int* rank);
```

See Also

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CGELSD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the minimum norm solution to a complex linear least squares problem: minimize $||b - A*x||$ using the singular value decomposition (SVD) of a rectangular matrix *A*, which may be rank-deficient.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

The effective rank of *A* equals the number of the singular values, which are greater than *rcond* multiplied by the largest singular value.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGELSD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CGELSD Example.
*  ==============
*
*  Program computes the minimum norm-solution to a complex linear least squares
*  problem using the singular value decomposition of A,
*  where A is the coefficient matrix:
*
```

```
*  (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
*  (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
*  ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)
*
*  and B is the right-hand side matrix:
*
*  ( -8.25,  7.98) (  2.91, -8.81)
*  ( -5.04,  3.33) (  6.19,  0.19)
*  (  7.98, -4.38) ( -5.96,  7.18)
*
*  Description.
*  ============
*
*  The routine computes the minimum-norm solution to a complex linear least
*  squares problem: minimize ||b - A*x|| using the singular value
*  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  The effective rank of A is determined by treating as zero those singular
*  values which are less than rcond times the largest singular value.
*
*  Example Program Results.
*  ========================
*
* CGELSD Example Program Results
*
* Minimum norm solution
* ( -0.08,  0.09) (  0.04,  0.16)
* ( -0.17,  0.10) (  0.17, -0.47)
* ( -0.92, -0.01) (  0.71, -0.41)
* ( -0.47, -0.26) (  0.69,  0.02)
*
* Effective rank =      3
*
* Singular values
*  20.01  18.21   7.88
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 3, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, RANK
      REAL             RCOND
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 3*MIN(M,N)*NLVL + 11*MIN(M,N),
*     RWORK dimension should be at least 10*MIN(M,N)+2*MIN(M,N)*SMLSIZ+
*     +8*MIN(M,N)*NLVL+3*SMLSIZ*NRHS+(SMLSIZ+1)**2,
*     where NLVL = MAX( 0, INT( LOG_2( MIN(M,N)/(SMLSIZ+1) ) )+1 )
```

```
*     and SMLSIZ = 25
      INTEGER          IWORK( 3*M*0+11*M )
      REAL             S( M ), RWORK( 10*M+2*M*25+8*M*0+3*25*NRHS+26*26
     $                 )
      COMPLEX          A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ ( 4.55,-0.32),( 8.87,-3.11),(-0.74, 1.16),
     $ (-4.36,-4.76),( 0.02, 8.43),( 3.80,-6.12),
     $ ( 3.99,-6.84),( 5.43,-9.30),(-7.24, 0.72),
     $ ( 8.03,-6.47),( 2.28, 8.94),( 2.21, 9.52)
     $                 /
      DATA             B/
     $ (-8.25, 7.98),(-5.04, 3.33),( 7.98,-4.38),( 0.00, 0.00),
     $ ( 2.91,-8.81),( 6.19, 0.19),(-5.96, 7.18),( 0.00, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CGELSD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CGELSD Example Program Results'
*     Negative RCOND means using default (machine precision) value
      RCOND = -1.0
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, RWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL CGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, RWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge;'
         WRITE(*,*)'the least squares solution could not be computed.'
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print effective rank.
*
      WRITE(*,'(/A,I6)')' Effective rank = ', RANK
*
*     Print singular values.
*
```

```
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
      STOP
      END
*
*     End of CGELSD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CGELSD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CGELSD Example.
   ==============

   Program computes the minimum norm-solution to a complex linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

   (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
   (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
   ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

   and B is the right-hand side matrix:

   ( -8.25,  7.98) (  2.91, -8.81)
   ( -5.04,  3.33) (  6.19,  0.19)
   (  7.98, -4.38) ( -5.96,  7.18)

   Description.
   ============

   The routine computes the minimum-norm solution to a complex linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 CGELSD Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
```

```
/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGELSD prototype */
extern void cgelsd( int* m, int* n, int* nrhs, fcomplex* a, int* lda,
                fcomplex* b, int* ldb, float* s, float* rcond, int* rank,
                fcomplex* work, int* lwork, float* rwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 3*min(m,n)*nlvl + 11*min(m,n),
                rwork dimension should be at least 10*min(m,n)+2*min(m,n)*smlsiz+
                +8*min(m,n)*nlvl+3*smlsiz*nrhs+(smlsiz+1)^2,
                where nlvl = max( 0, int( log_2( min(m,n)/(smlsiz+1) ) )+1 )
                and smlsiz = 25 */
        int iwork[3*M*0+11*M];
        float s[M], rwork[10*M+2*M*25+8*M*0+3*25*NRHS+26*26];
        fcomplex a[LDA*N] = {
           { 4.55f, -0.32f}, { 8.87f, -3.11f}, {-0.74f,  1.16f},
           {-4.36f, -4.76f}, { 0.02f,  8.43f}, { 3.80f, -6.12f},
           { 3.99f, -6.84f}, { 5.43f, -9.30f}, {-7.24f,  0.72f},
           { 8.03f, -6.47f}, { 2.28f,  8.94f}, { 2.21f,  9.52f}
        };
        fcomplex b[LDB*NRHS] = {
           {-8.25f,  7.98f}, {-5.04f,  3.33f}, { 7.98f, -4.38f}, { 0.00f,  0.00f},
           { 2.91f, -8.81f}, { 6.19f,  0.19f}, {-5.96f,  7.18f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
        printf( " CGELSD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, &wkopt, &lwork,
                        rwork, iwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve the equations A*X = B */
        cgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, work, &lwork,
                        rwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
```

```
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CGELSD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cgelsd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgelsd Example.
   =======================
```

```
  Program computes the minimum norm-solution to a complex linear least squares
  problem using the singular value decomposition of A,
  where A is the coefficient matrix:

  (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
  (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
  ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

  and B is the right-hand side matrix:

  ( -8.25,  7.98) (  2.91, -8.81)
  ( -5.04,  3.33) (  6.19,  0.19)
  (  7.98, -4.38) ( -5.96,  7.18)

  Description.
  ============

  The routine computes the minimum-norm solution to a complex linear least
  squares problem: minimize ||b - A*x|| using the singular value
  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  The effective rank of A is determined by treating as zero those singular
  values which are less than rcond times the largest singular value.

  Example Program Results.
  ========================

 LAPACKE_cgelsd (row-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA N
```

```
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        /* Local arrays */
        float s[M];
        MKL_Complex8 a[LDA*M] = {
           { 4.55f, -0.32f}, {-4.36f, -4.76f}, { 3.99f, -6.84f}, { 8.03f, -6.47f},
           { 8.87f, -3.11f}, { 0.02f,  8.43f}, { 5.43f, -9.30f}, { 2.28f,  8.94f},
           {-0.74f,  1.16f}, { 3.80f, -6.12f}, {-7.24f,  0.72f}, { 2.21f,  9.52f}
        };
        MKL_Complex8 b[LDB*N] = {
           {-8.25f,  7.98f}, { 2.91f, -8.81f},
           {-5.04f,  3.33f}, { 6.19f,  0.19f},
           { 7.98f, -4.38f}, {-5.96f,  7.18f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgelsd (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgelsd( LAPACK_ROW_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_cgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
```

```
                printf( "\n" );
        }
}
```

LAPACKE_cgelsd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgelsd Example.
   ======================

   Program computes the minimum norm-solution to a complex linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

   (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
   (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
   ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

   and B is the right-hand side matrix:

   ( -8.25,  7.98) (  2.91, -8.81)
   ( -5.04,  3.33) (  6.19,  0.19)
   (  7.98, -4.38) ( -5.96,  7.18)

   Description.
   ============

   The routine computes the minimum-norm solution to a complex linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================
```

```
 LAPACKE_cgelsd (column-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        /* Local arrays */
        float s[M];
        MKL_Complex8 a[LDA*N] = {
           { 4.55f, -0.32f}, { 8.87f, -3.11f}, {-0.74f,  1.16f},
           {-4.36f, -4.76f}, { 0.02f,  8.43f}, { 3.80f, -6.12f},
           { 3.99f, -6.84f}, { 5.43f, -9.30f}, {-7.24f,  0.72f},
           { 8.03f, -6.47f}, { 2.28f,  8.94f}, { 2.21f,  9.52f}
        };
        MKL_Complex8 b[LDB*NRHS] = {
           {-8.25f,  7.98f}, {-5.04f,  3.33f}, { 7.98f, -4.38f}, { 0.00f,  0.00f},
           { 2.91f, -8.81f}, { 6.19f,  0.19f}, {-5.96f,  7.18f}, { 0.00f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgelsd (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_cgelsd( LAPACK_COL_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
```

```
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_cgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

DGELSD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the minimum norm solution to a real linear least squares problem: minimize $||b - A*x||$ using the singular value decomposition (SVD) of a rectangular matrix *A*, which may be rank-deficient.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

The effective rank of *A* equals the number of the singular values, which are greater than *rcond* multiplied by the largest singular value.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout

- Example program in C for Column Major Data Layout

See Also

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGELSD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DGELSD Example.
*  ==============
*
*  Program computes the minimum norm-solution to a real linear least squares
*  problem using the singular value decomposition of A,
*  where A is the coefficient matrix:
*
*    0.12  -8.19   7.69  -2.26  -4.71
*   -6.91   2.22  -5.12  -9.08   9.96
*   -3.33  -8.94  -6.72  -4.40  -9.98
*    3.97   3.33  -2.74  -7.92  -3.20
*
*  and B is the right-hand side matrix:
*
*    7.30   0.47  -6.28
*    1.33   6.58  -3.42
*    2.68  -1.71   3.46
*   -9.62  -0.79   0.41
*
*  Description.
*  ============
*
*  The routine computes the minimum-norm solution to a real linear least
*  squares problem: minimize ||b - A*x|| using the singular value
*  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  The effective rank of A is determined by treating as zero those singular
*  values which are less than rcond times the largest singular value.
*
*  Example Program Results.
*  ========================
*
```

```
* DGELSD Example Program Results
*
* Minimum norm solution
*  -0.69  -0.24   0.06
*  -0.80  -0.08   0.21
*   0.38   0.12  -0.65
*   0.29  -0.24   0.42
*   0.29   0.35  -0.30
*
* Effective rank =      4
*
* Singular values
*  18.66  15.99  10.01   8.51
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 4, N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, RANK
      DOUBLE PRECISION RCOND
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 3*MIN(M,N)*NLVL + 11*MIN(M,N),
*     where NLVL = MAX( 0, INT( LOG_2( MIN(M,N)/(SMLSIZ+1) ) )+1 )
*     and SMLSIZ = 25
      INTEGER          IWORK( 3*M*0+11*M )
      DOUBLE PRECISION A( LDA, N ), B( LDB, NRHS ), S( M ),
     $                 WORK( LWMAX )
      DATA             A/
     $  0.12,-6.91,-3.33, 3.97,
     $ -8.19, 2.22,-8.94, 3.33,
     $  7.69,-5.12,-6.72,-2.74,
     $ -2.26,-9.08,-4.40,-7.92,
     $ -4.71, 9.96,-9.98,-3.20
     $                  /
      DATA             B/
     $  7.30, 1.33, 2.68,-9.62, 0.00,
     $  0.47, 6.58,-1.71,-0.79, 0.00,
     $ -6.28,-3.42, 3.46, 0.41, 0.00
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DGELSD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DGELSD Example Program Results'
*     Negative RCOND means using default (machine precision) value
      RCOND = -1.0
```

```
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL DGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge;'
         WRITE(*,*)'the least squares solution could not be computed.'
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print effective rank.
*
      WRITE(*,'(/A,I6)')' Effective rank = ', RANK
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, M, S, 1 )
      STOP
      END
*
*     End of DGELSD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DGELSD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DGELSD Example.
   ==============

   Program computes the minimum norm-solution to a real linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

     0.12  -8.19   7.69  -2.26  -4.71
    -6.91   2.22  -5.12  -9.08   9.96
    -3.33  -8.94  -6.72  -4.40  -9.98
     3.97   3.33  -2.74  -7.92  -3.20

   and B is the right-hand side matrix:

     7.30   0.47  -6.28
     1.33   6.58  -3.42
     2.68  -1.71   3.46
    -9.62  -0.79   0.41

   Description.
   ============

   The routine computes the minimum-norm solution to a real linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 DGELSD Example Program Results

 Minimum norm solution
```

```
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>

/* DGELSD prototype */
extern void dgelsd( int* m, int* n, int* nrhs, double* a, int* lda,
                double* b, int* ldb, double* s, double* rcond, int* rank,
                double* work, int* lwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 4
#define N 5
#define NRHS 3
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        double wkopt;
        double* work;
        /* Local arrays */
        /* iwork dimension should be at least 3*min(m,n)*nlvl + 11*min(m,n),
                where nlvl = max( 0, int( log_2( min(m,n)/(smlsiz+1) ) )+1 )
                and smlsiz = 25 */
        int iwork[3*M*0+11*M];
        double s[M];
        double a[LDA*N] = {
            0.12, -6.91, -3.33,  3.97,
           -8.19,  2.22, -8.94,  3.33,
            7.69, -5.12, -6.72, -2.74,
           -2.26, -9.08, -4.40, -7.92,
           -4.71,  9.96, -9.98, -3.20
        };
        double b[LDB*NRHS] = {
            7.30,  1.33,  2.68, -9.62,  0.00,
            0.47,  6.58, -1.71, -0.79,  0.00,
           -6.28, -3.42,  3.46,  0.41,  0.00
        };
        /* Executable statements */
        printf( " DGELSD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
```

```
        dgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, &wkopt, &lwork,
                        iwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve the equations A*X = B */
        dgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, work, &lwork,
                        iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DGELSD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dgelsd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgelsd Example.
   ======================

   Program computes the minimum norm-solution to a real linear least squares
```

```
  problem using the singular value decomposition of A,
  where A is the coefficient matrix:

    0.12  -8.19   7.69  -2.26  -4.71
   -6.91   2.22  -5.12  -9.08   9.96
   -3.33  -8.94  -6.72  -4.40  -9.98
    3.97   3.33  -2.74  -7.92  -3.20

  and B is the right-hand side matrix:

    7.30   0.47  -6.28
    1.33   6.58  -3.42
    2.68  -1.71   3.46
   -9.62  -0.79   0.41

  Description.
  ============

  The routine computes the minimum-norm solution to a real linear least
  squares problem: minimize ||b - A*x|| using the singular value
  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  The effective rank of A is determined by treating as zero those singular
  values which are less than rcond times the largest singular value.

  Example Program Results.
  ========================

 LAPACKE_dgelsd (row-major, high-level) Example Program Results

 Minimum norm solution
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 4
#define N 5
#define NRHS 3
#define LDA N
```

```
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        /* Local arrays */
        double s[M];
        double a[LDA*M] = {
            0.12, -8.19, 7.69, -2.26, -4.71,
           -6.91,  2.22, -5.12, -9.08,  9.96,
           -3.33, -8.94, -6.72, -4.40, -9.98,
            3.97,  3.33, -2.74, -7.92, -3.20
        };
        double b[LDB*N] = {
            7.30,  0.47, -6.28,
            1.33,  6.58, -3.42,
            2.68, -1.71, 3.46,
           -9.62, -0.79, 0.41,
            0.00,  0.00, 0.00
        };
        /* Executable statements */
        printf( "LAPACKE_dgelsd (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgelsd( LAPACK_ROW_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_dgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dgelsd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
```

```
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   LAPACKE_dgelsd Example.
   ======================

   Program computes the minimum norm-solution to a real linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

     0.12  -8.19   7.69  -2.26  -4.71
    -6.91   2.22  -5.12  -9.08   9.96
    -3.33  -8.94  -6.72  -4.40  -9.98
     3.97   3.33  -2.74  -7.92  -3.20

   and B is the right-hand side matrix:

     7.30   0.47  -6.28
     1.33   6.58  -3.42
     2.68  -1.71   3.46
    -9.62  -0.79   0.41

   Description.
   ============

   The routine computes the minimum-norm solution to a real linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 LAPACKE_dgelsd (column-major, high-level) Example Program Results

 Minimum norm solution
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
```

```
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 4
#define N 5
#define NRHS 3
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        /* Local arrays */
        double s[M];
        double a[LDA*N] = {
            0.12, -6.91, -3.33,  3.97,
           -8.19,  2.22, -8.94,  3.33,
            7.69, -5.12, -6.72, -2.74,
           -2.26, -9.08, -4.40, -7.92,
           -4.71,  9.96, -9.98, -3.20
        };
        double b[LDB*NRHS] = {
            7.30,  1.33,  2.68, -9.62,  0.00,
            0.47,  6.58, -1.71, -0.79,  0.00,
           -6.28, -3.42,  3.46,  0.41,  0.00
        };
        /* Executable statements */
        printf( "LAPACKE_dgelsd (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_dgelsd( LAPACK_COL_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
```

```
        exit( 0 );
} /* End of LAPACKE_dgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SGELSD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the minimum norm solution to a real linear least squares problem: minimize `||b – A*x||` using the singular value decomposition (SVD) of a rectangular matrix *A*, which may be rank-deficient.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

The effective rank of *A* equals the number of the singular values, which are greater than *rcond* multiplied by the largest singular value.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGELSD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SGELSD Example.
*  ==============
*
*  Program computes the minimum norm-solution to a real linear least squares
*  problem using the singular value decomposition of A,
*  where A is the coefficient matrix:
*
*    0.12  -8.19   7.69  -2.26  -4.71
*   -6.91   2.22  -5.12  -9.08   9.96
*   -3.33  -8.94  -6.72  -4.40  -9.98
*    3.97   3.33  -2.74  -7.92  -3.20
*
*  and B is the right-hand side matrix:
*
*    7.30   0.47  -6.28
*    1.33   6.58  -3.42
*    2.68  -1.71   3.46
*   -9.62  -0.79   0.41
*
*  Description.
*  ============
*
*  The routine computes the minimum-norm solution to a real linear least
*  squares problem: minimize ||b - A*x|| using the singular value
*  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  The effective rank of A is determined by treating as zero those singular
*  values which are less than rcond times the largest singular value.
*
*  Example Program Results.
*  ========================
*
* SGELSD Example Program Results
*
* Minimum norm solution
*  -0.69  -0.24   0.06
*  -0.80  -0.08   0.21
*   0.38   0.12  -0.65
*   0.29  -0.24   0.42
*   0.29   0.35  -0.30
*
* Effective rank =      4
*
* Singular values
*  18.66  15.99  10.01   8.51
*  =====================================================================
*
*     .. Parameters ..
```

```
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 4, N = 5, NRHS = 3 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, RANK
      REAL             RCOND
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 3*MIN(M,N)*NLVL + 11*MIN(M,N),
*     where NLVL = MAX( 0, INT( LOG_2( MIN(M,N)/(SMLSIZ+1) ) )+1 )
*     and SMLSIZ = 25
      INTEGER          IWORK( 3*M*0+11*M )
      REAL             A( LDA, N ), B( LDB, NRHS ), S( M ),
     $                 WORK( LWMAX )
      DATA             A/
     $  0.12,-6.91,-3.33, 3.97,
     $ -8.19, 2.22,-8.94, 3.33,
     $  7.69,-5.12,-6.72,-2.74,
     $ -2.26,-9.08,-4.40,-7.92,
     $ -4.71, 9.96,-9.98,-3.20
     $                  /
      DATA             B/
     $  7.30, 1.33, 2.68,-9.62, 0.00,
     $  0.47, 6.58,-1.71,-0.79, 0.00,
     $ -6.28,-3.42, 3.46, 0.41, 0.00
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SGELSD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SGELSD Example Program Results'
*     Negative RCOND means using default (machine precision) value
      RCOND = -1.0
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL SGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
```

```
         WRITE(*,*)'The algorithm computing SVD failed to converge;'
         WRITE(*,*)'the least squares solution could not be computed.'
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print effective rank.
*
      WRITE(*,'(/A,I6)')' Effective rank = ', RANK
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, M, S, 1 )
      STOP
      END
*
*     End of SGELSD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SGELSD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
```

```
*
*****************************************************************************
*/
/*
   SGELSD Example.
   ==============

   Program computes the minimum norm-solution to a real linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

     0.12  -8.19   7.69  -2.26  -4.71
    -6.91   2.22  -5.12  -9.08   9.96
    -3.33  -8.94  -6.72  -4.40  -9.98
     3.97   3.33  -2.74  -7.92  -3.20

   and B is the right-hand side matrix:

     7.30   0.47  -6.28
     1.33   6.58  -3.42
     2.68  -1.71   3.46
    -9.62  -0.79   0.41

   Description.
   ============

   The routine computes the minimum-norm solution to a real linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 SGELSD Example Program Results

 Minimum norm solution
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>

/* SGELSD prototype */
```

```
extern void sgelsd( int* m, int* n, int* nrhs, float* a, int* lda,
                float* b, int* ldb, float* s, float* rcond, int* rank,
                float* work, int* lwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 4
#define N 5
#define NRHS 3
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        float wkopt;
        float* work;
        /* Local arrays */
        /* iwork dimension should be at least 3*min(m,n)*nlvl + 11*min(m,n),
                where nlvl = max( 0, int( log_2( min(m,n)/(smlsiz+1) ) )+1 )
                and smlsiz = 25 */
        int iwork[3*M*0+11*M];
        float s[M];
        float a[LDA*N] = {
            0.12f, -6.91f, -3.33f,  3.97f,
           -8.19f,  2.22f, -8.94f,  3.33f,
            7.69f, -5.12f, -6.72f, -2.74f,
           -2.26f, -9.08f, -4.40f, -7.92f,
           -4.71f,  9.96f, -9.98f, -3.20f
        };
        float b[LDB*NRHS] = {
            7.30f,  1.33f,  2.68f, -9.62f,  0.00f,
            0.47f,  6.58f, -1.71f, -0.79f,  0.00f,
           -6.28f, -3.42f,  3.46f,  0.41f,  0.00f
        };
        /* Executable statements */
        printf( " SGELSD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        sgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, &wkopt, &lwork,
                        iwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve the equations A*X = B */
        sgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, work, &lwork,
                        iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
```

```
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SGELSD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_sgelsd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgelsd Example.
   =======================

   Program computes the minimum norm-solution to a real linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

     0.12  -8.19   7.69  -2.26  -4.71
    -6.91   2.22  -5.12  -9.08   9.96
    -3.33  -8.94  -6.72  -4.40  -9.98
     3.97   3.33  -2.74  -7.92  -3.20

   and B is the right-hand side matrix:

     7.30   0.47  -6.28
     1.33   6.58  -3.42
     2.68  -1.71   3.46
    -9.62  -0.79   0.41
```

```
   Description.
   ============

   The routine computes the minimum-norm solution to a real linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 LAPACKE_sgelsd (row-major, high-level) Example Program Results

 Minimum norm solution
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 4
#define N 5
#define NRHS 3
#define LDA N
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        /* Local arrays */
        float s[M];
        float a[LDA*M] = {
            0.12f, -8.19f, 7.69f, -2.26f, -4.71f,
           -6.91f,  2.22f, -5.12f, -9.08f,  9.96f,
           -3.33f, -8.94f, -6.72f, -4.40f, -9.98f,
            3.97f,  3.33f, -2.74f, -7.92f, -3.20f
```

```
        };
        float b[LDB*N] = {
            7.30f,  0.47f, -6.28f,
            1.33f,  6.58f, -3.42f,
            2.68f, -1.71f, 3.46f,
           -9.62f, -0.79f, 0.41f,
            0.00f,  0.00f, 0.00f
        };
        /* Executable statements */
        printf( "LAPACKE_sgelsd (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgelsd( LAPACK_ROW_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_sgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_sgelsd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
```

```
   LAPACKE_sgelsd Example.
   ======================

   Program computes the minimum norm-solution to a real linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

     0.12  -8.19   7.69  -2.26  -4.71
    -6.91   2.22  -5.12  -9.08   9.96
    -3.33  -8.94  -6.72  -4.40  -9.98
     3.97   3.33  -2.74  -7.92  -3.20

   and B is the right-hand side matrix:

     7.30   0.47  -6.28
     1.33   6.58  -3.42
     2.68  -1.71   3.46
    -9.62  -0.79   0.41

   Description.
   ============

   The routine computes the minimum-norm solution to a real linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 LAPACKE_sgelsd (column-major, high-level) Example Program Results

 Minimum norm solution
  -0.69  -0.24   0.06
  -0.80  -0.08   0.21
   0.38   0.12  -0.65
   0.29  -0.24   0.42
   0.29   0.35  -0.30

 Effective rank =      4

 Singular values
  18.66  15.99  10.01   8.51
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
```

```
#define M 4
#define N 5
#define NRHS 3
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        float rcond = -1.0;
        /* Local arrays */
        float s[M];
        float a[LDA*N] = {
            0.12f, -6.91f, -3.33f,  3.97f,
           -8.19f,  2.22f, -8.94f,  3.33f,
            7.69f, -5.12f, -6.72f, -2.74f,
           -2.26f, -9.08f, -4.40f, -7.92f,
           -4.71f,  9.96f, -9.98f, -3.20f
        };
        float b[LDB*NRHS] = {
            7.30f,  1.33f,  2.68f, -9.62f,  0.00f,
            0.47f,  6.58f, -1.71f, -0.79f,  0.00f,
           -6.28f, -3.42f,  3.46f,  0.41f,  0.00f
        };
        /* Executable statements */
        printf( "LAPACKE_sgelsd (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_sgelsd( LAPACK_COL_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_matrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_sgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZGELSD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the minimum norm solution to a complex linear least squares problem: minimize $||b - A*x||$ using the singular value decomposition (SVD) of a rectangular matrix *A*, which may be rank-deficient.

Several right-hand side vectors *b* and solution vectors *x* can be handled in a single call; they are stored as the columns of the right-hand side matrix *B* and the solution matrix *X*.

The effective rank of *A* equals the number of the singular values, which are greater than *rcond* multiplied by the largest singular value.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gelsd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGELSD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZGELSD Example.
*  ==============
*
*  Program computes the minimum norm-solution to a complex linear least squares
*  problem using the singular value decomposition of A,
*  where A is the coefficient matrix:
*
```

```
*  (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
*  (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
*  ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)
*
*  and B is the right-hand side matrix:
*
*  ( -8.25,  7.98) (  2.91, -8.81)
*  ( -5.04,  3.33) (  6.19,  0.19)
*  (  7.98, -4.38) ( -5.96,  7.18)
*
*  Description.
*  ============
*
*  The routine computes the minimum-norm solution to a complex linear least
*  squares problem: minimize ||b - A*x|| using the singular value
*  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.
*
*  Several right hand side vectors b and solution vectors x can be handled
*  in a single call; they are stored as the columns of the m-by-nrhs right
*  hand side matrix B and the n-by-nrhs solution matrix X.
*
*  The effective rank of A is determined by treating as zero those singular
*  values which are less than rcond times the largest singular value.
*
*  Example Program Results.
*  ========================
*
* ZGELSD Example Program Results
*
* Minimum norm solution
* ( -0.08,  0.09) (  0.04,  0.16)
* ( -0.17,  0.10) (  0.17, -0.47)
* ( -0.92, -0.01) (  0.71, -0.41)
* ( -0.47, -0.26) (  0.69,  0.02)
*
* Effective rank =      3
*
* Singular values
*  20.01  18.21   7.88
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N, NRHS
      PARAMETER        ( M = 3, N = 4, NRHS = 2 )
      INTEGER          LDA, LDB
      PARAMETER        ( LDA = M, LDB = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, RANK
      DOUBLE PRECISION RCOND
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 3*MIN(M,N)*NLVL + 11*MIN(M,N),
*     RWORK dimension should be at least 10*MIN(M,N)+2*MIN(M,N)*SMLSIZ+
*     +8*MIN(M,N)*NLVL+3*SMLSIZ*NRHS+(SMLSIZ+1)**2,
*     where NLVL = MAX( 0, INT( LOG_2( MIN(M,N)/(SMLSIZ+1) ) )+1 )
```

```
*     and SMLSIZ = 25
      INTEGER          IWORK( 3*M*0+11*M )
      DOUBLE PRECISION S( M ), RWORK( 10*M+2*M*25+8*M*0+3*25*NRHS+26*26
     $                 )
      COMPLEX*16       A( LDA, N ), B( LDB, NRHS ), WORK( LWMAX )
      DATA             A/
     $ ( 4.55,-0.32),( 8.87,-3.11),(-0.74, 1.16),
     $ (-4.36,-4.76),( 0.02, 8.43),( 3.80,-6.12),
     $ ( 3.99,-6.84),( 5.43,-9.30),(-7.24, 0.72),
     $ ( 8.03,-6.47),( 2.28, 8.94),( 2.21, 9.52)
     $                 /
      DATA             B/
     $ (-8.25, 7.98),(-5.04, 3.33),( 7.98,-4.38),( 0.00, 0.00),
     $ ( 2.91,-8.81),( 6.19, 0.19),(-5.96, 7.18),( 0.00, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZGELSD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZGELSD Example Program Results'
*     Negative RCOND means using default (machine precision) value
      RCOND = -1.0
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, RWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve the equations A*X = B.
*
      CALL ZGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK,
     $             LWORK, RWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge;'
         WRITE(*,*)'the least squares solution could not be computed.'
         STOP
      END IF
*
*     Print minimum norm solution.
*
      CALL PRINT_MATRIX( 'Minimum norm solution', N, NRHS, B, LDB )
*
*     Print effective rank.
*
      WRITE(*,'(/A,I6)')' Effective rank = ', RANK
*
*     Print singular values.
*
```

```
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
      STOP
      END
*
*     End of ZGELSD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZGELSD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZGELSD Example.
   ==============

   Program computes the minimum norm-solution to a complex linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

   (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
   (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
   ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

   and B is the right-hand side matrix:

   ( -8.25,  7.98) (  2.91, -8.81)
   ( -5.04,  3.33) (  6.19,  0.19)
   (  7.98, -4.38) ( -5.96,  7.18)

   Description.
   ============

   The routine computes the minimum-norm solution to a complex linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================

 ZGELSD Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
```

```
/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGELSD prototype */
extern void zgelsd( int* m, int* n, int* nrhs, dcomplex* a, int* lda,
                dcomplex* b, int* ldb, double* s, double* rcond, int* rank,
                dcomplex* work, int* lwork, double* rwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, lwork, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 3*min(m,n)*nlvl + 11*min(m,n),
                rwork dimension should be at least 10*min(m,n)+2*min(m,n)*smlsiz+
                +8*min(m,n)*nlvl+3*smlsiz*nrhs+(smlsiz+1)^2,
                where nlvl = max( 0, int( log_2( min(m,n)/(smlsiz+1) ) )+1 )
                and smlsiz = 25 */
        int iwork[3*M*0+11*M];
        double s[M], rwork[10*M+2*M*25+8*M*0+3*25*NRHS+26*26];
        dcomplex a[LDA*N] = {
           { 4.55, -0.32}, { 8.87, -3.11}, {-0.74,  1.16},
           {-4.36, -4.76}, { 0.02,  8.43}, { 3.80, -6.12},
           { 3.99, -6.84}, { 5.43, -9.30}, {-7.24,  0.72},
           { 8.03, -6.47}, { 2.28,  8.94}, { 2.21,  9.52}
        };
        dcomplex b[LDB*NRHS] = {
           {-8.25,  7.98}, {-5.04,  3.33}, { 7.98, -4.38}, { 0.00,  0.00},
           { 2.91, -8.81}, { 6.19,  0.19}, {-5.96,  7.18}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( " ZGELSD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, &wkopt, &lwork,
                        rwork, iwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve the equations A*X = B */
        zgelsd( &m, &n, &nrhs, a, &lda, b, &ldb, s, &rcond, &rank, work, &lwork,
                        rwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
```

```
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZGELSD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zgelsd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgelsd Example.
   =======================
```

```
  Program computes the minimum norm-solution to a complex linear least squares
  problem using the singular value decomposition of A,
  where A is the coefficient matrix:

  (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
  (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
  ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

  and B is the right-hand side matrix:

  ( -8.25,  7.98) (  2.91, -8.81)
  ( -5.04,  3.33) (  6.19,  0.19)
  (  7.98, -4.38) ( -5.96,  7.18)

  Description.
  ============

  The routine computes the minimum-norm solution to a complex linear least
  squares problem: minimize ||b - A*x|| using the singular value
  decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

  Several right hand side vectors b and solution vectors x can be handled
  in a single call; they are stored as the columns of the m-by-nrhs right
  hand side matrix B and the n-by-nrhs solution matrix X.

  The effective rank of A is determined by treating as zero those singular
  values which are less than rcond times the largest singular value.

  Example Program Results.
  ========================

 LAPACKE_zgelsd (row-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA N
```

```
#define LDB NRHS

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        /* Local arrays */
        double s[M];
        MKL_Complex16 a[LDA*M] = {
           { 4.55, -0.32}, {-4.36, -4.76}, { 3.99, -6.84}, { 8.03, -6.47},
           { 8.87, -3.11}, { 0.02,  8.43}, { 5.43, -9.30}, { 2.28,  8.94},
           {-0.74,  1.16}, { 3.80, -6.12}, {-7.24,  0.72}, { 2.21,  9.52}
        };
        MKL_Complex16 b[LDB*N] = {
           {-8.25,  7.98}, { 2.91, -8.81},
           {-5.04,  3.33}, { 6.19,  0.19},
           { 7.98, -4.38}, {-5.96,  7.18},
           { 0.00,  0.00}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zgelsd (row-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgelsd( LAPACK_ROW_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_zgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
```

```
                printf( "\n" );
        }
}
```

LAPACKE_zgelsd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgelsd Example.
   =======================

   Program computes the minimum norm-solution to a complex linear least squares
   problem using the singular value decomposition of A,
   where A is the coefficient matrix:

   (  4.55, -0.32) ( -4.36, -4.76) (  3.99, -6.84) (  8.03, -6.47)
   (  8.87, -3.11) (  0.02,  8.43) (  5.43, -9.30) (  2.28,  8.94)
   ( -0.74,  1.16) (  3.80, -6.12) ( -7.24,  0.72) (  2.21,  9.52)

   and B is the right-hand side matrix:

   ( -8.25,  7.98) (  2.91, -8.81)
   ( -5.04,  3.33) (  6.19,  0.19)
   (  7.98, -4.38) ( -5.96,  7.18)

   Description.
   ============

   The routine computes the minimum-norm solution to a complex linear least
   squares problem: minimize ||b - A*x|| using the singular value
   decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

   Several right hand side vectors b and solution vectors x can be handled
   in a single call; they are stored as the columns of the m-by-nrhs right
   hand side matrix B and the n-by-nrhs solution matrix X.

   The effective rank of A is determined by treating as zero those singular
   values which are less than rcond times the largest singular value.

   Example Program Results.
   ========================
```

```
 LAPACKE_zgelsd (column-major, high-level) Example Program Results

 Minimum norm solution
 ( -0.08,  0.09) (  0.04,  0.16)
 ( -0.17,  0.10) (  0.17, -0.47)
 ( -0.92, -0.01) (  0.71, -0.41)
 ( -0.47, -0.26) (  0.69,  0.02)

 Effective rank =      3

 Singular values
  20.01  18.21   7.88
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define NRHS 2
#define LDA M
#define LDB N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, nrhs = NRHS, lda = LDA, ldb = LDB, info, rank;
        /* Negative rcond means using default (machine precision) value */
        double rcond = -1.0;
        /* Local arrays */
        double s[M];
        MKL_Complex16 a[LDA*N] = {
           { 4.55, -0.32}, { 8.87, -3.11}, {-0.74,  1.16},
           {-4.36, -4.76}, { 0.02,  8.43}, { 3.80, -6.12},
           { 3.99, -6.84}, { 5.43, -9.30}, {-7.24,  0.72},
           { 8.03, -6.47}, { 2.28,  8.94}, { 2.21,  9.52}
        };
        MKL_Complex16 b[LDB*NRHS] = {
           {-8.25,  7.98}, {-5.04,  3.33}, { 7.98, -4.38}, { 0.00,  0.00},
           { 2.91, -8.81}, { 6.19,  0.19}, {-5.96,  7.18}, { 0.00,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zgelsd (column-major, high-level) Example Program Results\n" );
        /* Solve the equations A*X = B */
        info = LAPACKE_zgelsd( LAPACK_COL_MAJOR, m, n, nrhs, a, lda, b, ldb,
                        s, rcond, &rank );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge;\n" );
                printf( "the least squares solution could not be computed.\n" );
                exit( 1 );
        }
        /* Print minimum norm solution */
```

```
        print_matrix( "Minimum norm solution", n, nrhs, b, ldb );
        /* Print effective rank */
        printf( "\n Effective rank = %6i\n", rank );
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        exit( 0 );
} /* End of LAPACKE_zgelsd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

Symmetric Eigenproblems

Routine	Description	Examples
?syev	Computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.	dsyev ssyev
?heev	Computes all the eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.	cheev zheev
?syevd	Computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric matrix using a divide and conquer algorithm.	dsyevd ssyevd
?heevd	Computes all the eigenvalues and, optionally, all the eigenvectors of a complex Hermitian matrix using a divide and conquer algorithm.	cheevd zheevd
?syevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a symmetric matrix.	dsyevx ssyevx
?heevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.	cheevx zheevx
?syevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix using the Relatively Robust Representations.	dsyevr ssyevr

Routine	Description	Examples
?heevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix using the Relatively Robust Representations.	cheevr zheevr

?syev function

Computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call ssyev(jobz, uplo, n, a, lda, w, work, lwork, info)
call dsyev(jobz, uplo, n, a, lda, w, work, lwork, info)
```

Calling from C:

```
ssyev(jobz, uplo, n, a, lda, w, work, lwork, info);
dsyev(jobz, uplo, n, a, lda, w, work, lwork, info);
```

C Interface

```
lapack_int LAPACKE_ssyev(int matrix_layout, char jobz, char uplo, lapack_int n, float* a,
lapack_int lda, float* w);
lapack_int LAPACKE_dsyev(int matrix_layout, char jobz, char uplo, lapack_int n, double* a,
lapack_int lda, double* w);
```

See Also

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

DSYEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DSYEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DSYEV Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a real symmetric
*  matrix A:
*
*    1.96  -6.49  -0.47  -7.20  -0.65
*   -6.49   3.80  -6.39   1.50  -6.34
*   -0.47  -6.39   4.17  -1.51   2.67
*   -7.20   1.50  -1.51   5.70   1.80
*   -0.65  -6.34   2.67   1.80  -7.10
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*
```

```
*  Example Program Results.
*  ========================
*
* DSYEV Example Program Results
*
* Eigenvalues
* -11.07  -6.23   0.86   8.87  16.09
*
* Eigenvectors (stored columnwise)
*  -0.30  -0.61   0.40  -0.37   0.49
*  -0.51  -0.29  -0.41  -0.36  -0.61
*  -0.08  -0.38  -0.66   0.50   0.40
*   0.00  -0.45   0.46   0.62  -0.46
*  -0.80   0.45   0.17   0.31   0.16
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      DOUBLE PRECISION A( LDA, N ), W( N ), WORK( LWMAX )
      DATA             A/
     $  1.96, 0.00, 0.00, 0.00, 0.00,
     $ -6.49, 3.80, 0.00, 0.00, 0.00,
     $ -0.47,-6.39, 4.17, 0.00, 0.00,
     $ -7.20, 1.50,-1.51, 5.70, 0.00,
     $ -0.65,-6.34, 2.67, 1.80,-7.10
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DSYEV
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DSYEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DSYEV( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL DSYEV( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK, INFO )
*
*     Check for convergence.
```

```
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of DSYEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DSYEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
```

```
*/
/*
   DSYEV Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
    -0.47  -6.39   4.17  -1.51   2.67
    -7.20   1.50  -1.51   5.70   1.80
    -0.65  -6.34   2.67   1.80  -7.10

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 DSYEV Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>

/* DSYEV prototype */
extern void dsyev( char* jobz, char* uplo, int* n, double* a, int* lda,
                double* w, double* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork;
        double wkopt;
```

```
        double* work;
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            1.96,  0.00,  0.00,  0.00,  0.00,
           -6.49,  3.80,  0.00,  0.00,  0.00,
           -0.47, -6.39,  4.17,  0.00,  0.00,
           -7.20,  1.50, -1.51,  5.70,  0.00,
           -0.65, -6.34,  2.67,  1.80, -7.10
        };
        /* Executable statements */
        printf( " DSYEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dsyev( "Vectors", "Upper", &n, a, &lda, w, &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve eigenproblem */
        dsyev( "Vectors", "Upper", &n, a, &lda, w, work, &lwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DSYEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
```

```
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
    -0.47  -6.39   4.17  -1.51   2.67
    -7.20   1.50  -1.51   5.70   1.80
    -0.65  -6.34   2.67   1.80  -7.10

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_dsyev (row-major, high-level) Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
```

```
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            1.96, -6.49, -0.47, -7.20, -0.65,
            0.00,  3.80, -6.39,  1.50, -6.34,
            0.00,  0.00, 4.17, -1.51, 2.67,
            0.00,  0.00, 0.00,  5.70, 1.80,
            0.00,  0.00, 0.00,  0.00, -7.10
        };
        /* Executable statements */
        printf( "LAPACKE_dsyev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dsyev( LAPACK_ROW_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dsyev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyev Example.
   ======================
```

```
   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
    -0.47  -6.39   4.17  -1.51   2.67
    -7.20   1.50  -1.51   5.70   1.80
    -0.65  -6.34   2.67   1.80  -7.10

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_dsyev (column-major, high-level) Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            1.96,  0.00,  0.00,  0.00,  0.00,
           -6.49,  3.80,  0.00,  0.00,  0.00,
           -0.47, -6.39,  4.17,  0.00,  0.00,
           -7.20,  1.50, -1.51,  5.70,  0.00,
           -0.65, -6.34,  2.67,  1.80, -7.10
```

```
        };
        /* Executable statements */
        printf( "LAPACKE_dsyev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dsyev( LAPACK_COL_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dsyev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SSYEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout

- Example program in C for Column Major Data Layout

See Also

?syev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SSYEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SSYEV Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a real symmetric
*  matrix A:
*
*    1.96  -6.49  -0.47  -7.20  -0.65
*   -6.49   3.80  -6.39   1.50  -6.34
*   -0.47  -6.39   4.17  -1.51   2.67
*   -7.20   1.50  -1.51   5.70   1.80
*   -0.65  -6.34   2.67   1.80  -7.10
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*
*  Example Program Results.
*  ========================
*
* SSYEV Example Program Results
*
* Eigenvalues
* -11.07  -6.23   0.86   8.87  16.09
*
* Eigenvectors (stored columnwise)
*  -0.30  -0.61   0.40  -0.37   0.49
*  -0.51  -0.29  -0.41  -0.36  -0.61
*  -0.08  -0.38  -0.66   0.50   0.40
*   0.00  -0.45   0.46   0.62  -0.46
```

```
*  -0.80   0.45   0.17   0.31   0.16
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      REAL             A( LDA, N ), W( N ), WORK( LWMAX )
      DATA             A/
     $  1.96, 0.00, 0.00, 0.00, 0.00,
     $ -6.49, 3.80, 0.00, 0.00, 0.00,
     $ -0.47,-6.39, 4.17, 0.00, 0.00,
     $ -7.20, 1.50,-1.51, 5.70, 0.00,
     $ -0.65,-6.34, 2.67, 1.80,-7.10
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SSYEV
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SSYEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SSYEV( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL SSYEV( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
```

```
     $                   LDA )
      STOP
      END
*
*     End of SSYEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SSYEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SSYEV Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
    -0.47  -6.39   4.17  -1.51   2.67
    -7.20   1.50  -1.51   5.70   1.80
    -0.65  -6.34   2.67   1.80  -7.10
```

```
   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 SSYEV Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>

/* SSYEV prototype */
extern void ssyev( char* jobz, char* uplo, int* n, float* a, int* lda,
                float* w, float* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            1.96f,  0.00f,  0.00f,  0.00f,  0.00f,
           -6.49f,  3.80f,  0.00f,  0.00f,  0.00f,
           -0.47f, -6.39f,  4.17f,  0.00f,  0.00f,
           -7.20f,  1.50f, -1.51f,  5.70f,  0.00f,
           -0.65f, -6.34f,  2.67f,  1.80f, -7.10f
        };
        /* Executable statements */
        printf( " SSYEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
```

```
        lwork = -1;
        ssyev( "Vectors", "Upper", &n, a, &lda, w, &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve eigenproblem */
        ssyev( "Vectors", "Upper", &n, a, &lda, w, work, &lwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SSYEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
```

```
   -0.47  -6.39   4.17  -1.51   2.67
   -7.20   1.50  -1.51   5.70   1.80
   -0.65  -6.34   2.67   1.80  -7.10

  Description.
  ============

  The routine computes all eigenvalues and, optionally, eigenvectors of an
  n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

  A*v(j) = lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The computed eigenvectors are
  orthonormal.

  Example Program Results.
  ========================

 LAPACKE_ssyev (row-major, high-level) Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            1.96f, -6.49f, -0.47f, -7.20f, -0.65f,
            0.00f,  3.80f, -6.39f,  1.50f, -6.34f,
            0.00f,  0.00f, 4.17f, -1.51f, 2.67f,
            0.00f,  0.00f, 0.00f,  5.70f, 1.80f,
            0.00f,  0.00f, 0.00f,  0.00f, -7.10f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_ssyev( LAPACK_ROW_MAJOR, 'V', 'U', n, a, lda, w );
```

```
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_ssyev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A:

     1.96  -6.49  -0.47  -7.20  -0.65
    -6.49   3.80  -6.39   1.50  -6.34
    -0.47  -6.39   4.17  -1.51   2.67
    -7.20   1.50  -1.51   5.70   1.80
    -0.65  -6.34   2.67   1.80  -7.10

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
```

```
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_ssyev (column-major, high-level) Example Program Results

 Eigenvalues
 -11.07  -6.23   0.86   8.87  16.09

 Eigenvectors (stored columnwise)
  -0.30  -0.61   0.40  -0.37   0.49
  -0.51  -0.29  -0.41  -0.36  -0.61
  -0.08  -0.38  -0.66   0.50   0.40
   0.00  -0.45   0.46   0.62  -0.46
  -0.80   0.45   0.17   0.31   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            1.96f,  0.00f,  0.00f,  0.00f,  0.00f,
           -6.49f,  3.80f,  0.00f,  0.00f,  0.00f,
           -0.47f, -6.39f,  4.17f,  0.00f,  0.00f,
           -7.20f,  1.50f, -1.51f,  5.70f,  0.00f,
           -0.65f, -6.34f,  2.67f,  1.80f, -7.10f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_ssyev( LAPACK_COL_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
```

```
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_ssyev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?heev function

Computes all eigenvalues and, optionally, eigenvectors of a Hermitian matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call cheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
call zheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

Calling from C:

```
cheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info);
zheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info);
```

C Interface

```
lapack_int LAPACKE_cheev(int matrix_layout, char jobz, char uplo, lapack_int n,
lapack_complex_float* a, lapack_int lda, float* w);
lapack_int LAPACKE_zheev(int matrix_layout, char jobz, char uplo, lapack_int n,
lapack_complex_double* a, lapack_int lda, double* w);
```

See Also

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CHEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CHEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CHEEV Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a complex Hermitian
*  matrix A:
*
*  (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
*  ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
*  ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
*  ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
```

```
*  n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*
*  Example Program Results.
*  ========================
*
* CHEEV Example Program Results
*
* Eigenvalues
* -16.00  -6.76   6.67  25.51
*
* Eigenvectors (stored columnwise)
* (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
* (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
* ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
* (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least MAX(1,3*N-2)
      REAL             W( N ), RWORK( 3*N-2 )
      COMPLEX          A( LDA, N ), WORK( LWMAX )
      DATA             A/
     $ ( 9.14, 0.00),(-4.37, 9.22),(-1.98, 1.72),(-8.96, 9.50),
     $ ( 0.00, 0.00),(-3.35, 0.00),( 2.25, 9.51),( 2.57,-2.40),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-4.82, 0.00),(-3.24,-2.04),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 8.44, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CHEEV
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CHEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CHEEV( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
```

```
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL CHEEV( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $            INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of CHEEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
```

```
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CHEEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CHEEV Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 CHEEV Example Program Results

 Eigenvalues
```

```
 -16.00  -6.76   6.67  25.51

 Eigenvectors (stored columnwise)
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CHEEV prototype */
extern void cheev( char* jobz, char* uplo, int* n, fcomplex* a, int* lda,
                float* w, fcomplex* work, int* lwork, float* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least max(1,3*n-2) */
        float w[N], rwork[3*N-2];
        fcomplex a[LDA*N] = {
           { 9.14f,  0.00f}, {-4.37f,  9.22f}, {-1.98f,  1.72f}, {-8.96f,  9.50f},
           { 0.00f,  0.00f}, {-3.35f,  0.00f}, { 2.25f,  9.51f}, { 2.57f, -2.40f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-4.82f,  0.00f}, {-3.24f, -2.04f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 8.44f,  0.00f}
        };
        /* Executable statements */
        printf( " CHEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cheev( "Vectors", "Lower", &n, a, &lda, w, &wkopt, &lwork, rwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve eigenproblem */
        cheev( "Vectors", "Lower", &n, a, &lda, w, work, &lwork, rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
```

```
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CHEEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cheev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)
```

```
   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_cheev (row-major, high-level) Example Program Results

 Eigenvalues
 -16.00  -6.76   6.67  25.51

 Eigenvectors (stored columnwise)
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        MKL_Complex8 a[LDA*N] = {
           { 9.14f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-4.37f,  9.22f}, {-3.35f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-1.98f,  1.72f}, { 2.25f,  9.51f}, {-4.82f,  0.00f}, { 0.00f,  0.00f},
           {-8.96f,  9.50f}, { 2.57f, -2.40f}, {-3.24f, -2.04f}, { 8.44f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cheev( LAPACK_ROW_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
```

```
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cheev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cheev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)
```

```
  Description.
  ============

  The routine computes all eigenvalues and, optionally, eigenvectors of an
  n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

  A*v(j) = lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The computed eigenvectors are
  orthonormal.

  Example Program Results.
  ========================

 LAPACKE_cheev (column-major, high-level) Example Program Results

 Eigenvalues
 -16.00  -6.76   6.67  25.51

 Eigenvectors (stored columnwise)
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        MKL_Complex8 a[LDA*N] = {
           { 9.14f,  0.00f}, {-4.37f,  9.22f}, {-1.98f,  1.72f}, {-8.96f,  9.50f},
           { 0.00f,  0.00f}, {-3.35f,  0.00f}, { 2.25f,  9.51f}, { 2.57f, -2.40f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-4.82f,  0.00f}, {-3.24f, -2.04f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 8.44f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cheev( LAPACK_COL_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
```

```
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cheev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZHEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZHEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZHEEV Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a complex Hermitian
*  matrix A:
*
*  (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
*  ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
*  ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
*  ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*
*  Example Program Results.
*  ========================
*
* ZHEEV Example Program Results
*
* Eigenvalues
* -16.00  -6.76   6.67  25.51
*
* Eigenvectors (stored columnwise)
* (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
* (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
* ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
* (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
```

```
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least MAX(1,3*N-2)
      DOUBLE PRECISION W( N ), RWORK( 3*N-2 )
      COMPLEX*16       A( LDA, N ), WORK( LWMAX )
      DATA             A/
     $ ( 9.14, 0.00),(-4.37, 9.22),(-1.98, 1.72),(-8.96, 9.50),
     $ ( 0.00, 0.00),(-3.35, 0.00),( 2.25, 9.51),( 2.57,-2.40),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-4.82, 0.00),(-3.24,-2.04),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 8.44, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZHEEV
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZHEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZHEEV( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $            INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL ZHEEV( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $            INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
```

```
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of ZHEEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZHEEV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
```

```
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZHEEV Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 ZHEEV Example Program Results

 Eigenvalues
 -16.00  -6.76   6.67  25.51

 Eigenvectors (stored columnwise)
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZHEEV prototype */
extern void zheev( char* jobz, char* uplo, int* n, dcomplex* a, int* lda,
                double* w, dcomplex* work, int* lwork, double* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
```

```
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least max(1,3*n-2) */
        double w[N], rwork[3*N-2];
        dcomplex a[LDA*N] = {
           { 9.14,  0.00}, {-4.37,  9.22}, {-1.98,  1.72}, {-8.96,  9.50},
           { 0.00,  0.00}, {-3.35,  0.00}, { 2.25,  9.51}, { 2.57, -2.40},
           { 0.00,  0.00}, { 0.00,  0.00}, {-4.82,  0.00}, {-3.24, -2.04},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 8.44,  0.00}
        };
        /* Executable statements */
        printf( " ZHEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zheev( "Vectors", "Lower", &n, a, &lda, w, &wkopt, &lwork, rwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve eigenproblem */
        zheev( "Vectors", "Lower", &n, a, &lda, w, work, &lwork, rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZHEEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zheev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_zheev (row-major, high-level) Example Program Results

 Eigenvalues
 -16.00  -6.76   6.67  25.51

 Eigenvectors (stored columnwise)
```

```
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        MKL_Complex16 a[LDA*N] = {
           { 9.14,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-4.37,  9.22}, {-3.35,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-1.98,  1.72}, { 2.25,  9.51}, {-4.82,  0.00}, { 0.00,  0.00},
           {-8.96,  9.50}, { 2.57, -2.40}, {-3.24, -2.04}, { 8.44,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zheev( LAPACK_ROW_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zheev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
```

```
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_zheev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheev Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A:

   (  9.14,  0.00) ( -4.37, -9.22) ( -1.98, -1.72) ( -8.96, -9.50)
   ( -4.37,  9.22) ( -3.35,  0.00) (  2.25, -9.51) (  2.57,  2.40)
   ( -1.98,  1.72) (  2.25,  9.51) ( -4.82,  0.00) ( -3.24,  2.04)
   ( -8.96,  9.50) (  2.57, -2.40) ( -3.24, -2.04) (  8.44,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.

   Example Program Results.
   ========================

 LAPACKE_zheev (column-major, high-level) Example Program Results

 Eigenvalues
 -16.00  -6.76   6.67  25.51
```

```
 Eigenvectors (stored columnwise)
 (  0.34,  0.00) ( -0.55,  0.00) (  0.31,  0.00) ( -0.70,  0.00)
 (  0.44, -0.54) (  0.26,  0.18) (  0.45,  0.29) (  0.22, -0.28)
 ( -0.48, -0.37) ( -0.52, -0.02) ( -0.05,  0.57) (  0.15,  0.08)
 (  0.10, -0.12) ( -0.50,  0.28) ( -0.23, -0.48) (  0.34, -0.49)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        MKL_Complex16 a[LDA*N] = {
           { 9.14,  0.00}, {-4.37,  9.22}, {-1.98,  1.72}, {-8.96,  9.50},
           { 0.00,  0.00}, {-3.35,  0.00}, { 2.25,  9.51}, { 2.57, -2.40},
           { 0.00,  0.00}, { 0.00,  0.00}, {-4.82,  0.00}, {-3.24, -2.04},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 8.44,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zheev( LAPACK_COL_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zheev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
```

```
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?syevd function

Computes all eigenvalues and, optionally, all eigenvectors of a real symmetric matrix using divide and conquer algorithm.

NOTE
For a detailed description and reference information on this function, please visit:

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call ssyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
call dsyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
```

Calling from C:

```
ssyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info);
dsyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info);
```

C Interface

```
lapack_int LAPACKE_ssyevd(int matrix_layout, char jobz, char uplo, lapack_int n, float* a,
lapack_int lda, float* w);
lapack_int LAPACKE_dsyevd(int matrix_layout, char jobz, char uplo, lapack_int n, double* a,
lapack_int lda, double* w);
```

See Also

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

DSYEVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. This routine uses a divide and conquer algorithm to compute the eigenvalues and eigenvectors.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DSYEVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DSYEVD Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a real symmetric
*  matrix A using divide and conquer algorithm, where A is:
*
*    6.39   0.13  -8.23   5.71  -3.18
*    0.13   8.37  -4.46  -6.10   7.21
*   -8.23  -4.46  -9.58  -9.25  -7.42
*    5.71  -6.10  -9.25   3.72   8.54
*   -3.18   7.21  -7.42   8.54   2.51
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
```

```
*  n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  If the eigenvectors are requested, then this routine uses a divide and
*  conquer algorithm to compute eigenvalues and eigenvectors.
*
*  Example Program Results.
*  ========================
*
* DSYEVD Example Program Results
*
* Eigenvalues
* -17.44 -11.96   6.72  14.25  19.84
*
* Eigenvectors (stored columnwise)
*  -0.26   0.31  -0.74   0.33   0.42
*  -0.17  -0.39  -0.38  -0.80   0.16
*  -0.89   0.04   0.09   0.03  -0.45
*  -0.29  -0.59   0.34   0.31   0.60
*  -0.19   0.63   0.44  -0.38   0.48
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK
*
*     .. Local Arrays ..
      INTEGER          IWORK( LWMAX )
      DOUBLE PRECISION A( LDA, N ), W( N ), WORK( LWMAX )
      DATA             A/
     $  6.39, 0.00, 0.00, 0.00, 0.00,
     $  0.13, 8.37, 0.00, 0.00, 0.00,
     $ -8.23,-4.46,-9.58, 0.00, 0.00,
     $  5.71,-6.10,-9.25, 3.72, 0.00,
     $ -3.18, 7.21,-7.42, 8.54, 2.51
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DSYEVD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DSYEVD Example Program Results'
*
*     Query the optimal workspace.
```

```
*
      LWORK = -1
      LIWORK = -1
      CALL DSYEVD( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK,
     $             IWORK, LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL DSYEVD( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK,
     $             IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of DSYEVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DSYEVD Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   DSYEVD Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 DSYEVD Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84

 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
```

```
*/
#include <stdlib.h>
#include <stdio.h>

/* DSYEVD prototype */
extern void dsyevd( char* jobz, char* uplo, int* n, double* a, int* lda,
                double* w, double* work, int* lwork, int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork, liwork;
        int iwkopt;
        int* iwork;
        double wkopt;
        double* work;
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            6.39,  0.00,  0.00,  0.00,  0.00,
            0.13,  8.37,  0.00,  0.00,  0.00,
           -8.23, -4.46, -9.58,  0.00,  0.00,
            5.71, -6.10, -9.25,  3.72,  0.00,
           -3.18,  7.21, -7.42,  8.54,  2.51
        };
        /* Executable statements */
        printf( " DSYEVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        liwork = -1;
        dsyevd( "Vectors", "Upper", &n, a, &lda, w, &wkopt, &lwork, &iwkopt,
                        &liwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        dsyevd( "Vectors", "Upper", &n, a, &lda, w, work, &lwork, iwork,
                        &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)work );
        exit( 0 );
```

```
} /* End of DSYEVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
```

```
   ========================

 LAPACKE_dsyevd (row-major, high-level) Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84

 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            6.39,  0.13, -8.23, 5.71, -3.18,
            0.00,  8.37, -4.46, -6.10,  7.21,
            0.00,  0.00, -9.58, -9.25, -7.42,
            0.00,  0.00, 0.00, 3.72,  8.54,
            0.00,  0.00, 0.00, 0.00,  2.51
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevd (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dsyevd( LAPACK_ROW_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dsyevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
```

```
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_dsyevd (column-major, high-level) Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84
```

```
 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
*/
#include &lt;stdlib.h&gt;
#include &lt;stdio.h&gt;
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        double a[LDA*N] = {
            6.39,  0.00,  0.00,  0.00,  0.00,
            0.13,  8.37,  0.00,  0.00,  0.00,
           -8.23, -4.46, -9.58,  0.00,  0.00,
            5.71, -6.10, -9.25,  3.72,  0.00,
           -3.18,  7.21, -7.42,  8.54,  2.51
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevd (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dsyevd( LAPACK_COL_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info &gt; 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_dsyevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i &lt; m; i++ ) {
                for( j = 0; j &lt; n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SSYEVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where *lambda*`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. This routine uses a divide and conquer algorithm to compute the eigenvalues and eigenvectors.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SSYEVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SSYEVD Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a real symmetric
*  matrix A using divide and conquer algorithm, where A is:
*
```

```
*    6.39   0.13  -8.23   5.71  -3.18
*    0.13   8.37  -4.46  -6.10   7.21
*   -8.23  -4.46  -9.58  -9.25  -7.42
*    5.71  -6.10  -9.25   3.72   8.54
*   -3.18   7.21  -7.42   8.54   2.51
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  If the eigenvectors are requested, then this routine uses a divide and
*  conquer algorithm to compute eigenvalues and eigenvectors.
*
*  Example Program Results.
*  ========================
*
* SSYEVD Example Program Results
*
* Eigenvalues
* -17.44 -11.96   6.72  14.25  19.84
*
* Eigenvectors (stored columnwise)
*  -0.26   0.31  -0.74   0.33   0.42
*  -0.17  -0.39  -0.38  -0.80   0.16
*  -0.89   0.04   0.09   0.03  -0.45
*  -0.29  -0.59   0.34   0.31   0.60
*  -0.19   0.63   0.44  -0.38   0.48
*  ================================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK
*
*     .. Local Arrays ..
      INTEGER          IWORK( LWMAX )
      REAL             A( LDA, N ), W( N ), WORK( LWMAX )
      DATA             A/
     $  6.39, 0.00, 0.00, 0.00, 0.00,
     $  0.13, 8.37, 0.00, 0.00, 0.00,
     $ -8.23,-4.46,-9.58, 0.00, 0.00,
     $  5.71,-6.10,-9.25, 3.72, 0.00,
     $ -3.18, 7.21,-7.42, 8.54, 2.51
     $                  /
*
*     .. External Subroutines ..
```

```
      EXTERNAL         SSYEVD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SSYEVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      LIWORK = -1
      CALL SSYEVD( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK,
     $             IWORK, LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL SSYEVD( 'Vectors', 'Upper', N, A, LDA, W, WORK, LWORK,
     $             IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of SSYEVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
```

```
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SSYEVD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SSYEVD Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 SSYEVD Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84
```

```
 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
*/
#include <stdlib.h>
#include <stdio.h>

/* SSYEVD prototype */
extern void ssyevd( char* jobz, char* uplo, int* n, float* a, int* lda,
                float* w, float* work, int* lwork, int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork, liwork;
        int iwkopt;
        int* iwork;
        float wkopt;
        float* work;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            6.39f,  0.00f,  0.00f,  0.00f,  0.00f,
            0.13f,  8.37f,  0.00f,  0.00f,  0.00f,
           -8.23f, -4.46f, -9.58f,  0.00f,  0.00f,
            5.71f, -6.10f, -9.25f,  3.72f,  0.00f,
           -3.18f,  7.21f, -7.42f,  8.54f,  2.51f
        };
        /* Executable statements */
        printf( " SSYEVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        liwork = -1;
        ssyevd( "Vectors", "Upper", &n, a, &lda, w, &wkopt, &lwork, &iwkopt,
                        &liwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        ssyevd( "Vectors", "Upper", &n, a, &lda, w, work, &lwork, iwork,
                        &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
```

```
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)work );
        exit( 0 );
} /* End of SSYEVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)
```

```
   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_ssyevd (row-major, high-level) Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84

 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            6.39f,  0.13f, -8.23f, 5.71f, -3.18f,
            0.00f,  8.37f, -4.46f, -6.10f,  7.21f,
            0.00f,  0.00f, -9.58f, -9.25f, -7.42f,
            0.00f,  0.00f, 0.00f, 3.72f,  8.54f,
            0.00f,  0.00f, 0.00f, 0.00f,  2.51f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevd (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_ssyevd( LAPACK_ROW_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
```

```
        exit( 0 );
} /* End of LAPACKE_ssyevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a real symmetric
   matrix A using divide and conquer algorithm, where A is:

     6.39   0.13  -8.23   5.71  -3.18
     0.13   8.37  -4.46  -6.10   7.21
    -8.23  -4.46  -9.58  -9.25  -7.42
     5.71  -6.10  -9.25   3.72   8.54
    -3.18   7.21  -7.42   8.54   2.51

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.
```

```
   Example Program Results.
   ========================

 LAPACKE_ssyevd (column-major, high-level) Example Program Results

 Eigenvalues
 -17.44 -11.96   6.72  14.25  19.84

 Eigenvectors (stored columnwise)
  -0.26   0.31  -0.74   0.33   0.42
  -0.17  -0.39  -0.38  -0.80   0.16
  -0.89   0.04   0.09   0.03  -0.45
  -0.29  -0.59   0.34   0.31   0.60
  -0.19   0.63   0.44  -0.38   0.48
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        float a[LDA*N] = {
            6.39f,  0.00f,  0.00f,  0.00f,  0.00f,
            0.13f,  8.37f,  0.00f,  0.00f,  0.00f,
           -8.23f, -4.46f, -9.58f,  0.00f,  0.00f,
            5.71f, -6.10f, -9.25f,  3.72f,  0.00f,
           -3.18f,  7.21f, -7.42f,  8.54f,  2.51f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevd (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_ssyevd( LAPACK_COL_MAJOR, 'V', 'U', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_ssyevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?heevd function

Computes all eigenvalues and, optionally, all eigenvectors of a complex Hermitian matrix using divide and conquer algorithm.

NOTE
For a detailed description and reference information on this function, please visit:

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call cheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork, iwork, liwork, info)
call zheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork, iwork, liwork, info)
```

Calling from C:

```
cheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork, iwork, liwork, info);
zheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork, iwork, liwork, info);
```

C Interface

```
lapack_int LAPACKE_cheevd(int matrix_layout, char jobz, char uplo, lapack_int n,
lapack_complex_float* a, lapack_int lda, float* w);
lapack_int LAPACKE_zheevd(int matrix_layout, char jobz, char uplo, lapack_int n,
lapack_complex_double* a, lapack_int lda, double* w);
```

See Also

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CHEEVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda(j) is its eigenvalue.

The computed eigenvectors are orthonormal. This routine uses a divide and conquer algorithm to compute the eigenvalues and eigenvectors.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CHEEVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CHEEVD Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a complex Hermitian
*  matrix A using divide and conquer algorithm, where A is:
*
*  (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
*  ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
*  ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
*  (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
```

```
*  orthonormal.
*  If the eigenvectors are requested, then this routine uses a divide and
*  conquer algorithm to compute eigenvalues and eigenvectors.
*
*  Example Program Results.
*  ========================
*
* CHEEVD Example Program Results
*
* Eigenvalues
* -21.97  -0.05   6.46  16.34
*
* Eigenvectors (stored columnwise)
* (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
* (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
* (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
* ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK, LRWORK
*
*     .. Local Arrays ..
      INTEGER          IWORK( LWMAX )
      REAL             W( N ), RWORK( LWMAX )
      COMPLEX          A( LDA, N ), WORK( LWMAX )
      DATA             A/
     $ ( 3.40, 0.00),(-2.36, 1.93),(-4.68,-9.55),( 5.37, 1.23),
     $ ( 0.00, 0.00),( 6.94, 0.00),( 8.13, 1.47),( 2.07, 5.78),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-2.14, 0.00),( 4.68,-7.44),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),(-7.42, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CHEEVD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CHEEVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      LIWORK = -1
      LRWORK = -1
      CALL CHEEVD( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
```

```
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LRWORK = MIN( LWMAX, INT( RWORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL CHEEVD( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of CHEEVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
```

```
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CHEEVD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CHEEVD Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================
```

```
 CHEEVD Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CHEEVD prototype */
extern void cheevd( char* jobz, char* uplo, int* n, fcomplex* a, int* lda,
                float* w, fcomplex* work, int* lwork, float* rwork, int* lrwork,
                int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork, lrwork, liwork;
        int iwkopt;
        int* iwork;
        float rwkopt;
        float* rwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        float w[N];
        fcomplex a[LDA*N] = {
           { 3.40f,  0.00f}, {-2.36f,  1.93f}, {-4.68f, -9.55f}, { 5.37f,  1.23f},
           { 0.00f,  0.00f}, { 6.94f,  0.00f}, { 8.13f,  1.47f}, { 2.07f,  5.78f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-2.14f,  0.00f}, { 4.68f, -7.44f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-7.42f,  0.00f}
        };
        /* Executable statements */
        printf( " CHEEVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        lrwork = -1;
        liwork = -1;
        cheevd( "Vectors", "Lower", &n, a, &lda, w, &wkopt, &lwork, &rwkopt,
                        &lrwork, &iwkopt, &liwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
```

```
        lrwork = (int)rwkopt;
        rwork = (float*)malloc( lrwork*sizeof(float) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        cheevd( "Vectors", "Lower", &n, a, &lda, w, work, &lwork, rwork,
                        &lrwork, iwork, &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)rwork );
        free( (void*)work );
        exit( 0 );
} /* End of CHEEVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cheevd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
```

```
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_cheevd (row-major, high-level) Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
```

```
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        MKL_Complex8 a[LDA*N] = {
           { 3.40f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-2.36f,  1.93f}, { 6.94f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-4.68f, -9.55f}, { 8.13f,  1.47f}, {-2.14f,  0.00f}, { 0.00f,  0.00f},
           { 5.37f,  1.23f}, { 2.07f,  5.78f}, { 4.68f, -7.44f}, {-7.42f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevd (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cheevd( LAPACK_ROW_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cheevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cheevd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
```

```
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevd Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_cheevd (column-major, high-level) Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
```

```
/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        float w[N];
        MKL_Complex8 a[LDA*N] = {
           { 3.40f,  0.00f}, {-2.36f,  1.93f}, {-4.68f, -9.55f}, { 5.37f,  1.23f},
           { 0.00f,  0.00f}, { 6.94f,  0.00f}, { 8.13f,  1.47f}, { 2.07f,  5.78f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-2.14f,  0.00f}, { 4.68f, -7.44f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-7.42f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevd (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cheevd( LAPACK_COL_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_cheevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZHEEVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. This routine uses a divide and conquer algorithm to compute the eigenvalues and eigenvectors.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZHEEVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZHEEVD Example.
*  ==============
*
*  Program computes all eigenvalues and eigenvectors of a complex Hermitian
*  matrix A using divide and conquer algorithm, where A is:
```

```
*
*  (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
*  ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
*  ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
*  (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)
*
*  Description.
*  ============
*
*  The routine computes all eigenvalues and, optionally, eigenvectors of an
*  n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  If the eigenvectors are requested, then this routine uses a divide and
*  conquer algorithm to compute eigenvalues and eigenvectors.
*
*  Example Program Results.
*  ========================
*
* ZHEEVD Example Program Results
*
* Eigenvalues
* -21.97  -0.05   6.46  16.34
*
* Eigenvectors (stored columnwise)
* (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
* (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
* (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
* ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA
      PARAMETER        ( LDA = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK, LRWORK
*
*     .. Local Arrays ..
      INTEGER          IWORK( LWMAX )
      DOUBLE PRECISION W( N ), RWORK( LWMAX )
      COMPLEX*16       A( LDA, N ), WORK( LWMAX )
      DATA             A/
     $ ( 3.40, 0.00),(-2.36, 1.93),(-4.68,-9.55),( 5.37, 1.23),
     $ ( 0.00, 0.00),( 6.94, 0.00),( 8.13, 1.47),( 2.07, 5.78),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-2.14, 0.00),( 4.68,-7.44),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),(-7.42, 0.00)
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         ZHEEVD
```

```
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZHEEVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      LIWORK = -1
      LRWORK = -1
      CALL ZHEEVD( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LRWORK = MIN( LWMAX, INT( RWORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL ZHEEVD( 'Vectors', 'Lower', N, A, LDA, W, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Eigenvectors (stored columnwise)', N, N, A,
     $                   LDA )
      STOP
      END
*
*     End of ZHEEVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
```

```
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZHEEVD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZHEEVD Example.
   ==============

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
```

```
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 ZHEEVD Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZHEEVD prototype */
extern void zheevd( char* jobz, char* uplo, int* n, dcomplex* a, int* lda,
                double* w, dcomplex* work, int* lwork, double* rwork, int* lrwork,
                int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, info, lwork, lrwork, liwork;
        int iwkopt;
        int* iwork;
        double rwkopt;
        double* rwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        double w[N];
        dcomplex a[LDA*N] = {
```

```
           { 3.40,  0.00}, {-2.36,  1.93}, {-4.68, -9.55}, { 5.37,  1.23},
           { 0.00,  0.00}, { 6.94,  0.00}, { 8.13,  1.47}, { 2.07,  5.78},
           { 0.00,  0.00}, { 0.00,  0.00}, {-2.14,  0.00}, { 4.68, -7.44},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, {-7.42,  0.00}
        };
        /* Executable statements */
        printf( " ZHEEVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        lrwork = -1;
        liwork = -1;
        zheevd( "Vectors", "Lower", &n, a, &lda, w, &wkopt, &lwork, &rwkopt,
                        &lrwork, &iwkopt, &liwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        lrwork = (int)rwkopt;
        rwork = (double*)malloc( lrwork*sizeof(double) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        zheevd( "Vectors", "Lower", &n, a, &lda, w, work, &lwork, rwork,
                        &lrwork, iwork, &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)rwork );
        free( (void*)work );
        exit( 0 );
} /* End of ZHEEVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zheevd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheevd Example.
   ======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_zheevd (row-major, high-level) Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
```

```
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        MKL_Complex16 a[LDA*N] = {
           { 3.40,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-2.36,  1.93}, { 6.94,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-4.68, -9.55}, { 8.13,  1.47}, {-2.14,  0.00}, { 0.00,  0.00},
           { 5.37,  1.23}, { 2.07,  5.78}, { 4.68, -7.44}, {-7.42,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevd (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zheevd( LAPACK_ROW_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zheevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
```

```
                printf( "\n" );
        }
}
```

LAPACKE_zheevd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheevd Example.
   =======================

   Program computes all eigenvalues and eigenvectors of a complex Hermitian
   matrix A using divide and conquer algorithm, where A is:

   (  3.40,  0.00) ( -2.36, -1.93) ( -4.68,  9.55) (  5.37, -1.23)
   ( -2.36,  1.93) (  6.94,  0.00) (  8.13, -1.47) (  2.07, -5.78)
   ( -4.68, -9.55) (  8.13,  1.47) ( -2.14,  0.00) (  4.68,  7.44)
   (  5.37,  1.23) (  2.07,  5.78) (  4.68, -7.44) ( -7.42,  0.00)

   Description.
   ============

   The routine computes all eigenvalues and, optionally, eigenvectors of an
   n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   If the eigenvectors are requested, then this routine uses a divide and
   conquer algorithm to compute eigenvalues and eigenvectors.

   Example Program Results.
   ========================

 LAPACKE_zheevd (column-major, high-level) Example Program Results

 Eigenvalues
 -21.97  -0.05   6.46  16.34

 Eigenvectors (stored columnwise)
 (  0.41,  0.00) ( -0.34,  0.00) ( -0.69,  0.00) (  0.49,  0.00)
```

```
 (  0.02, -0.30) (  0.32, -0.21) ( -0.57, -0.22) ( -0.59, -0.21)
 (  0.18,  0.57) ( -0.42, -0.32) (  0.06,  0.16) ( -0.35, -0.47)
 ( -0.62, -0.09) ( -0.58,  0.35) ( -0.15, -0.31) ( -0.10, -0.12)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, info;
        /* Local arrays */
        double w[N];
        MKL_Complex16 a[LDA*N] = {
           { 3.40,  0.00}, {-2.36,  1.93}, {-4.68, -9.55}, { 5.37,  1.23},
           { 0.00,  0.00}, { 6.94,  0.00}, { 8.13,  1.47}, { 2.07,  5.78},
           { 0.00,  0.00}, { 0.00,  0.00}, {-2.14,  0.00}, { 4.68, -7.44},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, {-7.42,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevd (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zheevd( LAPACK_COL_MAJOR, 'V', 'L', n, a, lda, w );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_rmatrix( "Eigenvalues", 1, n, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Eigenvectors (stored columnwise)", n, n, a, lda );
        exit( 0 );
} /* End of LAPACKE_zheevd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?syevx function

Computes selected eigenvalues and, optionally, eigenvectors of a symmetric matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call ssyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work,
lwork, iwork, ifail, info)
call dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work,
lwork, iwork, ifail, info)
```

Calling from C:

```
ssyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work, lwork,
iwork, ifail, info);
dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work, lwork,
iwork, ifail, info);
```

C Interface

```
lapack_int LAPACKE_ssyevx(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, float* a, lapack_int lda, float vl, float vu, lapack_int il, lapack_int iu, float
abstol, lapack_int* m, float* w, float* z, lapack_int ldz, lapack_int* ifail);
lapack_int LAPACKE_dsyevx(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, double* a, lapack_int lda, double vl, double vu, lapack_int il, lapack_int iu, double
abstol, lapack_int* m, double* w, double* z, lapack_int ldz, lapack_int* ifail);
```

See Also

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

DSYEVX Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DSYEVX Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DSYEVX Example.
*  ==============
*
*  Program computes the smallest eigenvalues and the corresponding
*  eigenvectors of a real symmetric matrix A:
*
*    6.29  -0.39   0.61   1.18  -0.08
*   -0.39   7.19   0.81   1.19  -0.08
*    0.61   0.81   5.48  -3.13   0.22
*    1.18   1.19  -3.13   3.79  -0.26
*   -0.08  -0.08   0.22  -0.26   0.83
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
```

```
*  an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* DSYEVX Example Program Results
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.71   0.82   6.58
*
* Selected eigenvectors (stored columnwise)
*   0.22   0.09  -0.95
*   0.21   0.08  -0.04
*  -0.52  -0.22  -0.29
*  -0.73  -0.21  -0.09
*  -0.32   0.94   0.01
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N, NSELECT
      PARAMETER        ( N = 5, NSELECT = 3 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, IL, IU, M
      DOUBLE PRECISION ABSTOL, VL, VU
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 5*N
      INTEGER          IFAIL( N ), IWORK( 5*N )
      DOUBLE PRECISION A( LDA, N ), W( N ), Z( LDZ, NSELECT ),
     $                 WORK( LWMAX )
      DATA             A/
     $  6.29, 0.00, 0.00, 0.00, 0.00,
     $ -0.39, 7.19, 0.00, 0.00, 0.00,
     $  0.61, 0.81, 5.48, 0.00, 0.00,
     $  1.18, 1.19,-3.13, 3.79, 0.00,
     $ -0.08,-0.08, 0.22,-0.26, 0.83
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DSYEVX
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
```

```
*
*     .. Executable Statements ..
      WRITE(*,*)'DSYEVX Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set IL, IU to compute NSELECT smallest eigenvalues
      IL = 1
      IU = NSELECT
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DSYEVX( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL,
     $             INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL DSYEVX( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL,
     $             INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of DSYEVX Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
```

```
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DSYEVX Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DSYEVX Example.
   ==============

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:

     6.29  -0.39   0.61   1.18  -0.08
    -0.39   7.19   0.81   1.19  -0.08
     0.61   0.81   5.48  -3.13   0.22
     1.18   1.19  -3.13   3.79  -0.26
    -0.08  -0.08   0.22  -0.26   0.83

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================
```

```
 DSYEVX Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>

/* DSYEVX prototype */
extern void dsyevx( char* jobz, char* range, char* uplo, int* n, double* a,
                int* lda, double* vl, double* vu, int* il, int* iu, double* abstol,
                int* m, double* w, double* z, int* ldz, double* work, int* lwork,
      int* iwork, int* ifail, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, il, iu, m, lda = LDA, ldz = LDZ, info, lwork;
        double abstol, vl, vu;
        double wkopt;
        double* work;
        /* Local arrays */
   /* iwork dimension should be at least 5*n */
        int iwork[5*N], ifail[N];
        double w[N], z[LDZ*NSELECT];
        double a[LDA*N] = {
            6.29,  0.00,  0.00,  0.00,  0.00,
           -0.39,  7.19,  0.00,  0.00,  0.00,
            0.61,  0.81,  5.48,  0.00,  0.00,
            1.18,  1.19, -3.13,  3.79,  0.00,
           -0.08, -0.08,  0.22, -0.26,  0.83
        };
        /* Executable statements */
        printf( " DSYEVX Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Query and allocate the optimal workspace */
        lwork = -1;
```

```
        dsyevx( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, &wkopt, &lwork, iwork, ifail, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve eigenproblem */
        dsyevx( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, work, &lwork, iwork, ifail, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DSYEVX Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevx Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyevx Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:
```

```
    6.29  -0.39   0.61   1.18  -0.08
   -0.39   7.19   0.81   1.19  -0.08
    0.61   0.81   5.48  -3.13   0.22
    1.18   1.19  -3.13   3.79  -0.26
   -0.08  -0.08   0.22  -0.26   0.83

  Description.
  ============

  The routine computes selected eigenvalues and, optionally, eigenvectors of
  an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

  A*v(j) = lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The computed eigenvectors are
  orthonormal.
  Eigenvalues and eigenvectors can be selected by specifying either a range
  of values or a range of indices for the desired eigenvalues.

  Example Program Results.
  ========================

 LAPACKE_dsyevx (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ NSELECT

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        double w[N], z[LDZ*N];
```

```
        double a[LDA*N] = {
            6.29, -0.39,  0.61,  1.18, -0.08,
            0.00,  7.19,  0.81,  1.19, -0.08,
            0.00,  0.00,  5.48, -3.13, 0.22,
            0.00,  0.00,  0.00,  3.79, -0.26,
            0.00,  0.00,  0.00,  0.00, 0.83
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevx (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_dsyevx( LAPACK_ROW_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_dsyevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevx Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
```

```
*****************************************************************************
*/
/*
   LAPACKE_dsyevx Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:

     6.29  -0.39   0.61   1.18  -0.08
    -0.39   7.19   0.81   1.19  -0.08
     0.61   0.81   5.48  -3.13   0.22
     1.18   1.19  -3.13   3.79  -0.26
    -0.08  -0.08   0.22  -0.26   0.83

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_dsyevx (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N
```

```
/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        double w[N], z[LDZ*NSELECT];
        double a[LDA*N] = {
            6.29,  0.00,  0.00,  0.00,  0.00,
           -0.39,  7.19,  0.00,  0.00,  0.00,
            0.61,  0.81,  5.48,  0.00,  0.00,
            1.18,  1.19, -3.13,  3.79,  0.00,
           -0.08, -0.08,  0.22, -0.26,  0.83
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevx (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_dsyevx( LAPACK_COL_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_dsyevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SSYEVX Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SSYEVX Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SSYEVX Example.
*  ==============
*
*  Program computes the smallest eigenvalues and the corresponding
*  eigenvectors of a real symmetric matrix A:
*
*    6.29  -0.39   0.61   1.18  -0.08
*   -0.39   7.19   0.81   1.19  -0.08
*    0.61   0.81   5.48  -3.13   0.22
*    1.18   1.19  -3.13   3.79  -0.26
*   -0.08  -0.08   0.22  -0.26   0.83
*
*  Description.
*  ============
```

```
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* SSYEVX Example Program Results
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.71   0.82   6.58
*
* Selected eigenvectors (stored columnwise)
*   0.22   0.09  -0.95
*   0.21   0.08  -0.04
*  -0.52  -0.22  -0.29
*  -0.73  -0.21  -0.09
*  -0.32   0.94   0.01
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N, NSELECT
      PARAMETER        ( N = 5, NSELECT = 3 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, IL, IU, M
      REAL             ABSTOL, VL, VU
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 5*N
      INTEGER          IFAIL( N ), IWORK( 5*N )
      REAL             A( LDA, N ), W( N ), Z( LDZ, NSELECT ),
     $                 WORK( LWMAX )
      DATA             A/
     $  6.29, 0.00, 0.00, 0.00, 0.00,
     $ -0.39, 7.19, 0.00, 0.00, 0.00,
     $  0.61, 0.81, 5.48, 0.00, 0.00,
     $  1.18, 1.19,-3.13, 3.79, 0.00,
     $ -0.08,-0.08, 0.22,-0.26, 0.83
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SSYEVX
      EXTERNAL         PRINT_MATRIX
*
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SSYEVX Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set IL, IU to compute NSELECT smallest eigenvalues
      IL = 1
      IU = NSELECT
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SSYEVX( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL,
     $             INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL SSYEVX( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL,
     $             INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of SSYEVX Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
```

```
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SSYEVX Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   SSYEVX Example.
   ==============

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:

     6.29  -0.39   0.61   1.18  -0.08
    -0.39   7.19   0.81   1.19  -0.08
     0.61   0.81   5.48  -3.13   0.22
     1.18   1.19  -3.13   3.79  -0.26
    -0.08  -0.08   0.22  -0.26   0.83

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
```

```
   ========================

 SSYEVX Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>

/* SSYEVX prototype */
extern void ssyevx( char* jobz, char* range, char* uplo, int* n, float* a,
                int* lda, float* vl, float* vu, int* il, int* iu, float* abstol,
                int* m, float* w, float* z, int* ldz, float* work, int* lwork,
      int* iwork, int* ifail, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, il, iu, m, lda = LDA, ldz = LDZ, info, lwork;
        float abstol, vl, vu;
        float wkopt;
        float* work;
        /* Local arrays */
   /* iwork dimension should be at least 5*n */
        int iwork[5*N], ifail[N];
        float w[N], z[LDZ*NSELECT];
        float a[LDA*N] = {
            6.29f,  0.00f,  0.00f,  0.00f,  0.00f,
           -0.39f,  7.19f,  0.00f,  0.00f,  0.00f,
            0.61f,  0.81f,  5.48f,  0.00f,  0.00f,
            1.18f,  1.19f, -3.13f,  3.79f,  0.00f,
           -0.08f, -0.08f,  0.22f, -0.26f,  0.83f
        };
        /* Executable statements */
        printf( " SSYEVX Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
```

```
        /* Query and allocate the optimal workspace */
        lwork = -1;
        ssyevx( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, &wkopt, &lwork, iwork, ifail, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve eigenproblem */
        ssyevx( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, work, &lwork, iwork, ifail, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SSYEVX Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevx Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevx Example.
   ======================
```

```
   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:

     6.29  -0.39   0.61   1.18  -0.08
    -0.39   7.19   0.81   1.19  -0.08
     0.61   0.81   5.48  -3.13   0.22
     1.18   1.19  -3.13   3.79  -0.26
    -0.08  -0.08   0.22  -0.26   0.83

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_ssyevx (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ NSELECT

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        float abstol, vl, vu;
        /* Local arrays */
```

```
        MKL_INT ifail[N];
        float w[N], z[LDZ*N];
        float a[LDA*N] = {
            6.29f, -0.39f,  0.61f,  1.18f, -0.08f,
            0.00f,  7.19f,  0.81f,  1.19f, -0.08f,
            0.00f,  0.00f,  5.48f, -3.13f, 0.22f,
            0.00f,  0.00f,  0.00f,  3.79f, -0.26f,
            0.00f,  0.00f,  0.00f,  0.00f, 0.83f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevx (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_ssyevx( LAPACK_ROW_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_ssyevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevx Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
```

```
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevx Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A:

     6.29  -0.39   0.61   1.18  -0.08
    -0.39   7.19   0.81   1.19  -0.08
     0.61   0.81   5.48  -3.13   0.22
     1.18   1.19  -3.13   3.79  -0.26
    -0.08  -0.08   0.22  -0.26   0.83

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_ssyevx (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.71   0.82   6.58

 Selected eigenvectors (stored columnwise)
   0.22   0.09  -0.95
   0.21   0.08  -0.04
  -0.52  -0.22  -0.29
  -0.73  -0.21  -0.09
  -0.32   0.94   0.01
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
```

```
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        float w[N], z[LDZ*NSELECT];
        float a[LDA*N] = {
            6.29f,  0.00f,  0.00f,  0.00f,  0.00f,
           -0.39f,  7.19f,  0.00f,  0.00f,  0.00f,
            0.61f,  0.81f,  5.48f,  0.00f,  0.00f,
            1.18f,  1.19f, -3.13f,  3.79f,  0.00f,
           -0.08f, -0.08f,  0.22f, -0.26f,  0.83f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevx (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_ssyevx( LAPACK_COL_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_ssyevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?heevx function

Computes selected eigenvalues and, optionally, eigenvectors of a Hermitian matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call cheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work,
lwork, rwork, iwork, ifail, info)
call zheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work,
lwork, rwork, iwork, ifail, info)
```

Calling from C:

```
cheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work, lwork,
rwork, iwork, ifail, info);
zheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work, lwork,
rwork, iwork, ifail, info);
```

C Interface

```
lapack_int LAPACKE_cheevx(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, lapack_complex_float* a, lapack_int lda, float vl, float vu, lapack_int il, lapack_int
iu, float abstol, lapack_int* m, float* w, lapack_complex_float* z, lapack_int ldz,
lapack_int* ifail);
lapack_int LAPACKE_zheevx(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, lapack_complex_double* a, lapack_int lda, double vl, double vu, lapack_int il,
lapack_int iu, double abstol, lapack_int* m, double* w, lapack_complex_double* z,
lapack_int ldz, lapack_int* ifail);
```

See Also

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CHEEVX Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the selected eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CHEEVX Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CHEEVX Example.
*  ==============
*
*  Program computes eigenvalues specified by a selected range of values
*  and corresponding eigenvectors of a complex Hermitian matrix A:
*
*  (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
*  ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
*  ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
*  (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
```

```
*
* CHEEVX Example Program Results
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.09   9.53  18.75
*
* Selected eigenvectors (stored columnwise)
* (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
* ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
* (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
* ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, IL, IU, M
      REAL             ABSTOL, VL, VU
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 5*N
      INTEGER          IWORK( 5*N ), IFAIL( N )
*     RWORK dimension should be at least 7*N
      REAL             W( N ), RWORK( 7*N )
      COMPLEX          A( LDA, N ), Z( LDZ, N ), WORK( LWMAX )
      DATA             A/
     $ ( 6.51, 0.00),(-5.92,-9.53),(-2.46,-2.91),( 8.84,-3.21),
     $ ( 0.00, 0.00),(-1.73, 0.00),( 6.50,-2.09),( 1.32,-8.81),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 6.90, 0.00),(-0.59,-2.47),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),(-2.85, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CHEEVX
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CHEEVX Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set VL, VU to compute eigenvalues in half-open (VL,VU] interval
      VL = 0.0
      VU = 100.0
*
*     Query the optimal workspace.
*
      LWORK = -1
```

```
      CALL CHEEVX( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK,
     $             IFAIL, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL CHEEVX( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK,
     $             IFAIL, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of CHEEVX Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
```

```
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CHEEVX Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   CHEEVX Example.
   ==============

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
```

```
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 CHEEVX Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CHEEVX prototype */
extern void cheevx( char* jobz, char* range, char* uplo, int* n, fcomplex* a,
                int* lda, float* vl, float* vu, int* il, int* iu, float* abstol,
                int* m, float* w, fcomplex* z, int* ldz, fcomplex* work, int* lwork,
                float* rwork, int* iwork, int* ifail, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldz = LDZ, il, iu, m, info, lwork;
        float abstol, vl, vu;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 5*n */
        int iwork[5*N], ifail[N];
        /* rwork dimension should be at least 7*n */
        float w[N], rwork[7*N];
        fcomplex z[LDZ*N];
        fcomplex a[LDA*N] = {
           { 6.51f,  0.00f}, {-5.92f, -9.53f}, {-2.46f, -2.91f}, { 8.84f, -3.21f},
           { 0.00f,  0.00f}, {-1.73f,  0.00f}, { 6.50f, -2.09f}, { 1.32f, -8.81f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 6.90f,  0.00f}, {-0.59f, -2.47f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-2.85f,  0.00f}
```

```
        };
        /* Executable statements */
        printf( " CHEEVX Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cheevx( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, &wkopt, &lwork, rwork, iwork, ifail, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve eigenproblem */
        cheevx( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, work, &lwork, rwork, iwork, ifail, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CHEEVX Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cheevx Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevx Example.
   ======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_cheevx (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
```

```
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        float w[N];
        MKL_Complex8 z[LDZ*N];
        MKL_Complex8 a[LDA*N] = {
           { 6.51f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-5.92f, -9.53f}, {-1.73f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-2.46f, -2.91f}, { 6.50f, -2.09f}, { 6.90f,  0.00f}, { 0.00f,  0.00f},
           { 8.84f, -3.21f}, { 1.32f, -8.81f}, {-0.59f, -2.47f}, {-2.85f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevx (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Solve eigenproblem */
        info = LAPACKE_cheevx( LAPACK_ROW_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_cheevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cheevx Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevx Example.
   =======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
```

```
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_cheevx (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        float w[N];
        MKL_Complex8 z[LDZ*N];
        MKL_Complex8 a[LDA*N] = {
           { 6.51f,  0.00f}, {-5.92f, -9.53f}, {-2.46f, -2.91f}, { 8.84f, -3.21f},
           { 0.00f,  0.00f}, {-1.73f,  0.00f}, { 6.50f, -2.09f}, { 1.32f, -8.81f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 6.90f,  0.00f}, {-0.59f, -2.47f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-2.85f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevx (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Solve eigenproblem */
        info = LAPACKE_cheevx( LAPACK_COL_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
```

```
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_cheevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZHEEVX Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the selected eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran

- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevx function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZHEEVX Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZHEEVX Example.
*  ==============
*
*  Program computes eigenvalues specified by a selected range of values
*  and corresponding eigenvectors of a complex Hermitian matrix A:
*
*  (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
*  ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
*  ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
*  (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* ZHEEVX Example Program Results
```

```
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.09   9.53  18.75
*
* Selected eigenvectors (stored columnwise)
* (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
* ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
* (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
* ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, IL, IU, M
      DOUBLE PRECISION ABSTOL, VL, VU
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 5*N
      INTEGER          IWORK( 5*N ), IFAIL( N )
*     RWORK dimension should be at least 7*N
      DOUBLE PRECISION W( N ), RWORK( 7*N )
      COMPLEX*16       A( LDA, N ), Z( LDZ, N ), WORK( LWMAX )
      DATA             A/
     $ ( 6.51, 0.00),(-5.92,-9.53),(-2.46,-2.91),( 8.84,-3.21),
     $ ( 0.00, 0.00),(-1.73, 0.00),( 6.50,-2.09),( 1.32,-8.81),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 6.90, 0.00),(-0.59,-2.47),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),(-2.85, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZHEEVX
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZHEEVX Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set VL, VU to compute eigenvalues in half-open (VL,VU] interval
      VL = 0.0
      VU = 100.0
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZHEEVX( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK,
```

```
     $             IFAIL, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL ZHEEVX( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK,
     $             IFAIL, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of ZHEEVX Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
```

```
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZHEEVX Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   ZHEEVX Example.
   ==============

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.
```

```
  Example Program Results.
  ========================

 ZHEEVX Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZHEEVX prototype */
extern void zheevx( char* jobz, char* range, char* uplo, int* n, dcomplex* a,
                int* lda, double* vl, double* vu, int* il, int* iu, double* abstol,
                int* m, double* w, dcomplex* z, int* ldz, dcomplex* work, int* lwork,
                double* rwork, int* iwork, int* ifail, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldz = LDZ, il, iu, m, info, lwork;
        double abstol, vl, vu;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 5*n */
        int iwork[5*N], ifail[N];
        /* rwork dimension should be at least 7*n */
        double w[N], rwork[7*N];
        dcomplex z[LDZ*N];
        dcomplex a[LDA*N] = {
           { 6.51,  0.00}, {-5.92, -9.53}, {-2.46, -2.91}, { 8.84, -3.21},
           { 0.00,  0.00}, {-1.73,  0.00}, { 6.50, -2.09}, { 1.32, -8.81},
           { 0.00,  0.00}, { 0.00,  0.00}, { 6.90,  0.00}, {-0.59, -2.47},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, {-2.85,  0.00}
        };
        /* Executable statements */
```

```
        printf( " ZHEEVX Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zheevx( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, &wkopt, &lwork, rwork, iwork, ifail, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve eigenproblem */
        zheevx( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, work, &lwork, rwork, iwork, ifail, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZHEEVX Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zheevx Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
```

```
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
********************************************************************************
*/
/*
   LAPACKE_zheevx Example.
   ======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_zheevx (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"
```

```
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        double w[N];
        MKL_Complex16 z[LDZ*N];
        MKL_Complex16 a[LDA*N] = {
           { 6.51,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-5.92, -9.53}, {-1.73,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-2.46, -2.91}, { 6.50, -2.09}, { 6.90,  0.00}, { 0.00,  0.00},
           { 8.84, -3.21}, { 1.32, -8.81}, {-0.59, -2.47}, {-2.85,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevx (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Solve eigenproblem */
        info = LAPACKE_zheevx( LAPACK_ROW_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_zheevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

```
/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_zheevx Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheevx Example.
   =======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A:

   (  6.51,  0.00) ( -5.92,  9.53) ( -2.46,  2.91) (  8.84,  3.21)
   ( -5.92, -9.53) ( -1.73,  0.00) (  6.50,  2.09) (  1.32,  8.81)
   ( -2.46, -2.91) (  6.50, -2.09) (  6.90,  0.00) ( -0.59,  2.47)
   (  8.84, -3.21) (  1.32, -8.81) ( -0.59, -2.47) ( -2.85,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================
```

```
 LAPACKE_zheevx (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.09   9.53  18.75

 Selected eigenvectors (stored columnwise)
 (  0.18,  0.00) ( -0.54,  0.00) (  0.67,  0.00)
 ( -0.40, -0.31) ( -0.21, -0.17) ( -0.30, -0.43)
 (  0.60,  0.40) ( -0.35, -0.28) ( -0.39, -0.34)
 ( -0.34,  0.26) ( -0.57,  0.35) (  0.05,  0.05)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT ifail[N];
        double w[N];
        MKL_Complex16 z[LDZ*N];
        MKL_Complex16 a[LDA*N] = {
           { 6.51,  0.00}, {-5.92, -9.53}, {-2.46, -2.91}, { 8.84, -3.21},
           { 0.00,  0.00}, {-1.73,  0.00}, { 6.50, -2.09}, { 1.32, -8.81},
           { 0.00,  0.00}, { 0.00,  0.00}, { 6.90,  0.00}, {-0.59, -2.47},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, {-2.85,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevx (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = 0.0;
        vu = 100.0;
        /* Solve eigenproblem */
        info = LAPACKE_zheevx( LAPACK_COL_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, ifail );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
```

```
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_zheevx Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?syevr function

Computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix using the Relatively Robust Representations.

NOTE
For a detailed description and reference information on this function, please visit:

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call ssyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
work, lwork, iwork, liwork, info)
call dsyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
work, lwork, iwork, liwork, info)
```

Calling from C:

```
ssyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz, work,
lwork, iwork, liwork, info);
dsyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz, work,
lwork, iwork, liwork, info);
```

C Interface

```
lapack_int LAPACKE_ssyevr(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, float* a, lapack_int lda, float vl, float vu, lapack_int il, lapack_int iu, float
abstol, lapack_int* m, float* w, float* z, lapack_int ldz, lapack_int* isuppz);
```

```
lapack_int LAPACKE_dsyevr(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, double* a, lapack_int lda, double vl, double vu, lapack_int il, lapack_int iu, double
abstol, lapack_int* m, double* w, double* z, lapack_int ldz, lapack_int* isuppz);
```

See Also

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

DSYEVR Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DSYEVR Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DSYEVR Example.
*  ==============
*
*  Program computes the smallest eigenvalues and the corresponding
*  eigenvectors of a real symmetric matrix A using the Relatively Robust
*  Representations, where A is:
*
*    0.67  -0.20   0.19  -1.06   0.46
*   -0.20   3.82  -0.13   1.06  -0.48
*    0.19  -0.13   3.27   0.11   1.10
*   -1.06   1.06   0.11   5.86  -0.98
*    0.46  -0.48   1.10  -0.98   3.54
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* DSYEVR Example Program Results
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.43   2.14   3.37
*
* Selected eigenvectors (stored columnwise)
*  -0.98  -0.01  -0.08
*   0.01   0.02  -0.93
*   0.04  -0.69  -0.07
*  -0.18   0.19   0.31
*   0.07   0.69  -0.13
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          N, NSELECT
      PARAMETER        ( N = 5, NSELECT = 3 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
```

```
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK, IL, IU, M
      DOUBLE PRECISION ABSTOL, VL, VU
*
*     .. Local Arrays ..
      INTEGER          ISUPPZ( N ), IWORK( LWMAX )
      DOUBLE PRECISION A( LDA, N ), W( N ), Z( LDZ, NSELECT ),
     $                 WORK( LWMAX )
      DATA             A/
     $  0.67, 0.00, 0.00, 0.00, 0.00,
     $ -0.20, 3.82, 0.00, 0.00, 0.00,
     $  0.19,-0.13, 3.27, 0.00, 0.00,
     $ -1.06, 1.06, 0.11, 5.86, 0.00,
     $  0.46,-0.48, 1.10,-0.98, 3.54
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DSYEVR
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DSYEVR Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set IL, IU to compute NSELECT smallest eigenvalues
      IL = 1
      IU = NSELECT
*
*     Query the optimal workspace.
*
      LWORK = -1
      LIWORK = -1
      CALL DSYEVR( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
     $             LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL DSYEVR( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
     $             LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
```

```
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of DSYEVR Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DSYEVR Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DSYEVR Example.
   ==============

   Program computes the smallest eigenvalues and the corresponding
```

```
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:

     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 DSYEVR Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>

/* DSYEVR prototype */
extern void dsyevr( char* jobz, char* range, char* uplo, int* n, double* a,
                int* lda, double* vl, double* vu, int* il, int* iu, double* abstol,
                int* m, double* w, double* z, int* ldz, int* isuppz, double* work,
                int* lwork, int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
```

```
        /* Locals */
        int n = N, il, iu, m, lda = LDA, ldz = LDZ, info, lwork, liwork;
        double abstol, vl, vu;
        int iwkopt;
        int* iwork;
        double wkopt;
        double* work;
        /* Local arrays */
        int isuppz[N];
        double w[N], z[LDZ*NSELECT];
        double a[LDA*N] = {
            0.67,  0.00,  0.00,  0.00,  0.00,
           -0.20,  3.82,  0.00,  0.00,  0.00,
            0.19, -0.13,  3.27,  0.00,  0.00,
           -1.06,  1.06,  0.11,  5.86,  0.00,
            0.46, -0.48,  1.10, -0.98,  3.54
        };
        /* Executable statements */
        printf( " DSYEVR Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        liwork = -1;
        dsyevr( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, &wkopt, &lwork, &iwkopt, &liwork,
                        &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        dsyevr( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, work, &lwork, iwork, &liwork,
                        &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)work );
        exit( 0 );
} /* End of DSYEVR Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevr Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyevr Example.
   ======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:

     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_dsyevr (row-major, high-level) Example Program Results
```

```
 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ NSELECT

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        double w[N], z[LDZ*N];
        double a[LDA*N] = {
            0.67, -0.20, 0.19, -1.06, 0.46,
            0.00,  3.82, -0.13,  1.06, -0.48,
            0.00,  0.00, 3.27,  0.11, 1.10,
            0.00,  0.00, 0.00,  5.86, -0.98,
            0.00,  0.00, 0.00,  0.00, 3.54
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevr (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_dsyevr( LAPACK_ROW_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
```

```
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_dsyevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dsyevr Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dsyevr Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:

     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
```

```
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_dsyevr (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        double w[N], z[LDZ*NSELECT];
        double a[LDA*N] = {
            0.67,  0.00,  0.00,  0.00,  0.00,
           -0.20,  3.82,  0.00,  0.00,  0.00,
            0.19, -0.13,  3.27,  0.00,  0.00,
           -1.06,  1.06,  0.11,  5.86,  0.00,
            0.46, -0.48,  1.10, -0.98,  3.54
        };
        /* Executable statements */
        printf( "LAPACKE_dsyevr (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_dsyevr( LAPACK_COL_MAJOR, 'V', 'I', 'U', n, a, lda,
```

```
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_dsyevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SSYEVR Example

NOTE
For a detailed description and reference information on this function, please visit:

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes all the eigenvalues and, optionally, the eigenvectors of a square real symmetric matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - C
?syevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SSYEVR Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SSYEVR Example.
*  ==============
*
*  Program computes the smallest eigenvalues and the corresponding
*  eigenvectors of a real symmetric matrix A using the Relatively Robust
*  Representations, where A is:
*
*    0.67  -0.20   0.19  -1.06   0.46
*   -0.20   3.82  -0.13   1.06  -0.48
*    0.19  -0.13   3.27   0.11   1.10
*   -1.06   1.06   0.11   5.86  -0.98
*    0.46  -0.48   1.10  -0.98   3.54
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* SSYEVR Example Program Results
*
* The total number of eigenvalues found: 3
*
* Selected eigenvalues
*   0.43   2.14   3.37
*
* Selected eigenvectors (stored columnwise)
```

```
*  -0.98  -0.01  -0.08
*   0.01   0.02  -0.93
*   0.04  -0.69  -0.07
*  -0.18   0.19   0.31
*   0.07   0.69  -0.13
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          N, NSELECT
      PARAMETER        ( N = 5, NSELECT = 3 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LIWORK, IL, IU, M
      REAL             ABSTOL, VL, VU
*
*     .. Local Arrays ..
      INTEGER          ISUPPZ( N ), IWORK( LWMAX )
      REAL             A( LDA, N ), W( N ), Z( LDZ, NSELECT ),
     $                 WORK( LWMAX )
      DATA             A/
     $  0.67, 0.00, 0.00, 0.00, 0.00,
     $ -0.20, 3.82, 0.00, 0.00, 0.00,
     $  0.19,-0.13, 3.27, 0.00, 0.00,
     $ -1.06, 1.06, 0.11, 5.86, 0.00,
     $  0.46,-0.48, 1.10,-0.98, 3.54
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SSYEVR
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SSYEVR Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set IL, IU to compute NSELECT smallest eigenvalues
      IL = 1
      IU = NSELECT
*
*     Query the optimal workspace.
*
      LWORK = -1
      LIWORK = -1
      CALL SSYEVR( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
     $             LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
```

```
      CALL SSYEVR( 'Vectors', 'Indices', 'Upper', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
     $             LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of SSYEVR Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SSYEVR Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
```

```
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SSYEVR Example.
   ==============

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:

     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 SSYEVR Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>

/* SSYEVR prototype */
```

```
extern void ssyevr( char* jobz, char* range, char* uplo, int* n, float* a,
                int* lda, float* vl, float* vu, int* il, int* iu, float* abstol,
                int* m, float* w, float* z, int* ldz, int* isuppz, float* work,
                int* lwork, int* iwork, int* liwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, il, iu, m, lda = LDA, ldz = LDZ, info, lwork, liwork;
        float abstol, vl, vu;
        int iwkopt;
        int* iwork;
        float wkopt;
        float* work;
        /* Local arrays */
        int isuppz[N];
        float w[N], z[LDZ*NSELECT];
        float a[LDA*N] = {
            0.67f,  0.00f,  0.00f,  0.00f,  0.00f,
           -0.20f,  3.82f,  0.00f,  0.00f,  0.00f,
            0.19f, -0.13f,  3.27f,  0.00f,  0.00f,
           -1.06f,  1.06f,  0.11f,  5.86f,  0.00f,
            0.46f, -0.48f,  1.10f, -0.98f,  3.54f
        };
        /* Executable statements */
        printf( " SSYEVR Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        liwork = -1;
        ssyevr( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, &wkopt, &lwork, &iwkopt, &liwork,
                        &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        ssyevr( "Vectors", "Indices", "Upper", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, work, &lwork, iwork, &liwork,
                        &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
```

```
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)work );
        exit( 0 );
} /* End of SSYEVR Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevr Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevr Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:

     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============
```

```
   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_ssyevr (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ NSELECT

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        float w[N], z[LDZ*N];
        float a[LDA*N] = {
            0.67f, -0.20f, 0.19f, -1.06f, 0.46f,
            0.00f,  3.82f, -0.13f,  1.06f, -0.48f,
            0.00f,  0.00f, 3.27f,  0.11f, 1.10f,
            0.00f,  0.00f, 0.00f,  5.86f, -0.98f,
            0.00f,  0.00f, 0.00f,  0.00f, 3.54f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevr (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
```

```
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_ssyevr( LAPACK_ROW_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_ssyevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_ssyevr Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_ssyevr Example.
   =======================

   Program computes the smallest eigenvalues and the corresponding
   eigenvectors of a real symmetric matrix A using the Relatively Robust
   Representations, where A is:
```

```
     0.67  -0.20   0.19  -1.06   0.46
    -0.20   3.82  -0.13   1.06  -0.48
     0.19  -0.13   3.27   0.11   1.10
    -1.06   1.06   0.11   5.86  -0.98
     0.46  -0.48   1.10  -0.98   3.54

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n real symmetric matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_ssyevr (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 3

 Selected eigenvalues
   0.43   2.14   3.37

 Selected eigenvectors (stored columnwise)
  -0.98  -0.01  -0.08
   0.01   0.02  -0.93
   0.04  -0.69  -0.07
  -0.18   0.19   0.31
   0.07   0.69  -0.13
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 5
#define NSELECT 3
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, il, iu, m, lda = LDA, ldz = LDZ, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        float w[N], z[LDZ*NSELECT];
        float a[LDA*N] = {
```

```
            0.67f,  0.00f,  0.00f,  0.00f,  0.00f,
           -0.20f,  3.82f,  0.00f,  0.00f,  0.00f,
            0.19f, -0.13f,  3.27f,  0.00f,  0.00f,
           -1.06f,  1.06f,  0.11f,  5.86f,  0.00f,
            0.46f, -0.48f,  1.10f, -0.98f,  3.54f
        };
        /* Executable statements */
        printf( "LAPACKE_ssyevr (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set il, iu to compute NSELECT smallest eigenvalues */
        il = 1;
        iu = NSELECT;
        /* Solve eigenproblem */
        info = LAPACKE_ssyevr( LAPACK_COL_MAJOR, 'V', 'I', 'U', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_matrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_ssyevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?heevr function

Computes selected eigenvalues and, optionally, eigenvectors of a Hermitian matrix using the Relatively Robust Representations.

NOTE
For a detailed description and reference information on this function, please visit:

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call cheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
work, lwork, rwork, lrwork, iwork, liwork, info)
call zheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
work, lwork, rwork, lrwork, iwork, liwork, info)
```

Calling from C:

```
cheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz, work,
lwork, rwork, lrwork, iwork, liwork, info);
zheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz, work,
lwork, rwork, lrwork, iwork, liwork, info);
```

C Interface

```
lapack_int LAPACKE_cheevr(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, lapack_complex_float* a, lapack_int lda, float vl, float vu, lapack_int il, lapack_int
iu, float abstol, lapack_int* m, float* w, lapack_complex_float* z, lapack_int ldz,
lapack_int* isuppz);
lapack_int LAPACKE_zheevr(int matrix_layout, char jobz, char range, char uplo, lapack_int
n, lapack_complex_double* a, lapack_int lda, double vl, double vu, lapack_int il,
lapack_int iu, double abstol, lapack_int* m, double* w, lapack_complex_double* z,
lapack_int ldz, lapack_int* isuppz);
```

See Also

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CHEEVR Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the selected eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout

- Example program in C for Column Major Data Layout

See Also

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C
?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CHEEVR Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CHEEVR Example.
*  ==============
*
*  Program computes eigenvalues specified by a selected range of values
*  and corresponding eigenvectors of a complex Hermitian matrix A using the
*  Relatively Robust Representations, where A is:
*
*  ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
*  ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
*  ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
*  ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* CHEEVR Example Program Results
*
* The total number of eigenvalues found: 2
*
* Selected eigenvalues
*  -4.18   3.57
*
* Selected eigenvectors (stored columnwise)
```

```
* (  0.68,  0.00) (  0.38,  0.00)
* (  0.03,  0.18) (  0.54, -0.57)
* ( -0.03,  0.21) ( -0.40,  0.04)
* (  0.20,  0.64) ( -0.14, -0.26)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LRWORK, LIWORK, IL, IU, M
      REAL             ABSTOL, VL, VU
*
*     .. Local Arrays ..
      INTEGER          ISUPPZ( N ), IWORK( LWMAX )
      REAL             W( N ), RWORK( LWMAX )
      COMPLEX          A( LDA, N ), Z( LDZ, N ), WORK( LWMAX )
      DATA             A/
     $ (-2.16, 0.00),(-0.16, 4.86),(-7.23, 9.38),(-0.04,-6.86),
     $ ( 0.00, 0.00),( 7.45, 0.00),( 4.39,-6.29),(-8.11, 4.41),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-9.03, 0.00),(-6.89, 7.66),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 7.76, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         CHEEVR
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CHEEVR Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set VL, VU to compute eigenvalues in half-open (VL,VU] interval
      VL = -5.0
      VU = 5.0
*
*     Query the optimal workspace.
*
      LWORK = -1
      LRWORK = -1
      LIWORK = -1
      CALL CHEEVR( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LRWORK = MIN( LWMAX, INT( RWORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
```

```
      CALL CHEEVR( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of CHEEVR Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
```

```
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CHEEVR Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CHEEVR Example.
   ==============

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================
```

```
 CHEEVR Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57

 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CHEEVR prototype */
extern void cheevr( char* jobz, char* range, char* uplo, int* n, fcomplex* a,
                int* lda, float* vl, float* vu, int* il, int* iu, float* abstol,
                int* m, float* w, fcomplex* z, int* ldz, int* isuppz, fcomplex* work,
                int* lwork, float* rwork, int* lrwork, int* iwork, int* liwork,
                int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldz = LDZ, il, iu, m, info, lwork, lrwork, liwork;
        float abstol, vl, vu;
        int iwkopt;
        int* iwork;
        float rwkopt;
        float* rwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        int isuppz[N];
        float w[N];
        fcomplex z[LDZ*N];
        fcomplex a[LDA*N] = {
           {-2.16f,  0.00f}, {-0.16f,  4.86f}, {-7.23f,  9.38f}, {-0.04f, -6.86f},
           { 0.00f,  0.00f}, { 7.45f,  0.00f}, { 4.39f, -6.29f}, {-8.11f,  4.41f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-9.03f,  0.00f}, {-6.89f,  7.66f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 7.76f,  0.00f}
        };
        /* Executable statements */
        printf( " CHEEVR Example Program Results\n" );
```

```
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = -5.0;
        vu = 5.0;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        lrwork = -1;
        liwork = -1;
        cheevr( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, &wkopt, &lwork, &rwkopt, &lrwork,
                        &iwkopt, &liwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        lrwork = (int)rwkopt;
        rwork = (float*)malloc( lrwork*sizeof(float) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        cheevr( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, work, &lwork, rwork, &lrwork,
                        iwork, &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)rwork );
        free( (void*)work );
        exit( 0 );
} /* End of CHEEVR Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
```

```
                printf( "\n" );
        }
}
```

LAPACKE_cheevr Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevr Example.
   =======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_cheevr (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57
```

```
 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        float w[N];
        MKL_Complex8 z[LDZ*N];
        MKL_Complex8 a[LDA*N] = {
           {-2.16f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-0.16f,  4.86f}, { 7.45f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f},
           {-7.23f,  9.38f}, { 4.39f, -6.29f}, {-9.03f,  0.00f}, { 0.00f,  0.00f},
           {-0.04f, -6.86f}, {-8.11f,  4.41f}, {-6.89f,  7.66f}, { 7.76f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevr (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = -5.0;
        vu = 5.0;
        /* Solve eigenproblem */
        info = LAPACKE_cheevr( LAPACK_ROW_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_cheevr Example */
```

```
/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cheevr Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cheevr Example.
   =======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
```

```
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_cheevr (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57

 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        float abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        float w[N];
        MKL_Complex8 z[LDZ*N];
        MKL_Complex8 a[LDA*N] = {
           {-2.16f,  0.00f}, {-0.16f,  4.86f}, {-7.23f,  9.38f}, {-0.04f, -6.86f},
           { 0.00f,  0.00f}, { 7.45f,  0.00f}, { 4.39f, -6.29f}, {-8.11f,  4.41f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, {-9.03f,  0.00f}, {-6.89f,  7.66f},
           { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 0.00f,  0.00f}, { 7.76f,  0.00f}
        };
        /* Executable statements */
        printf( "LAPACKE_cheevr (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
```

```
        vl = -5.0;
        vu = 5.0;
        /* Solve eigenproblem */
        info = LAPACKE_cheevr( LAPACK_COL_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_cheevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

ZHEEVR Example

NOTE
For a detailed description and reference information on this function, please visit:

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the selected eigenvalues and, optionally, the eigenvectors of a square complex Hermitian matrix *A*.

The eigenvector `v(j)` of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

lambda`(j)` is its eigenvalue.

The computed eigenvectors are orthonormal. The eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices of the desired eigenvalues.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - C

?heevr function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZHEEVR Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZHEEVR Example.
*  ==============
*
*  Program computes eigenvalues specified by a selected range of values
*  and corresponding eigenvectors of a complex Hermitian matrix A using the
*  Relatively Robust Representations, where A is:
*
*  ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
*  ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
*  ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
*  ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)
*
*  Description.
*  ============
*
*  The routine computes selected eigenvalues and, optionally, eigenvectors of
*  an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies
*
*  A*v(j) = lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The computed eigenvectors are
*  orthonormal.
*  Eigenvalues and eigenvectors can be selected by specifying either a range
```

```
*  of values or a range of indices for the desired eigenvalues.
*
*  Example Program Results.
*  ========================
*
* ZHEEVR Example Program Results
*
* The total number of eigenvalues found: 2
*
* Selected eigenvalues
*  -4.18   3.57
*
* Selected eigenvectors (stored columnwise)
* (  0.68,  0.00) (  0.38,  0.00)
* (  0.03,  0.18) (  0.54, -0.57)
* ( -0.03,  0.21) ( -0.40,  0.04)
* (  0.20,  0.64) ( -0.14, -0.26)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDZ
      PARAMETER        ( LDA = N, LDZ = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK, LRWORK, LIWORK, IL, IU, M
      DOUBLE PRECISION ABSTOL, VL, VU
*
*     .. Local Arrays ..
      INTEGER          ISUPPZ( N ), IWORK( LWMAX )
      DOUBLE PRECISION W( N ), RWORK( LWMAX )
      COMPLEX*16       A( LDA, N ), Z( LDZ, N ), WORK( LWMAX )
      DATA             A/
     $ (-2.16, 0.00),(-0.16, 4.86),(-7.23, 9.38),(-0.04,-6.86),
     $ ( 0.00, 0.00),( 7.45, 0.00),( 4.39,-6.29),(-8.11, 4.41),
     $ ( 0.00, 0.00),( 0.00, 0.00),(-9.03, 0.00),(-6.89, 7.66),
     $ ( 0.00, 0.00),( 0.00, 0.00),( 0.00, 0.00),( 7.76, 0.00)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZHEEVR
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZHEEVR Example Program Results'
*     Negative ABSTOL means using the default value
      ABSTOL = -1.0
*     Set VL, VU to compute eigenvalues in half-open (VL,VU] interval
      VL = -5.0
      VU = 5.0
*
*     Query the optimal workspace.
```

```
*
      LWORK = -1
      LRWORK = -1
      LIWORK = -1
      CALL ZHEEVR( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
      LRWORK = MIN( LWMAX, INT( RWORK( 1 ) ) )
      LIWORK = MIN( LWMAX, IWORK( 1 ) )
*
*     Solve eigenproblem.
*
      CALL ZHEEVR( 'Vectors', 'Values', 'Lower', N, A, LDA, VL, VU, IL,
     $             IU, ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK,
     $             LRWORK, IWORK, LIWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print the number of eigenvalues found.
*
      WRITE(*,'(/A,I2)')' The total number of eigenvalues found:', M
*
*     Print eigenvalues.
*
      CALL PRINT_RMATRIX( 'Selected eigenvalues', 1, M, W, 1 )
*
*     Print eigenvectors.
*
      CALL PRINT_MATRIX( 'Selected eigenvectors (stored columnwise)',
     $                   N, M, Z, LDZ )
      STOP
      END
*
*     End of ZHEEVR Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
```

```
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZHEEVR Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZHEEVR Example.
   ==============

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============
```

```
   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 ZHEEVR Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57

 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZHEEVR prototype */
extern void zheevr( char* jobz, char* range, char* uplo, int* n, dcomplex* a,
                int* lda, double* vl, double* vu, int* il, int* iu, double* abstol,
                int* m, double* w, dcomplex* z, int* ldz, int* isuppz, dcomplex* work,
                int* lwork, double* rwork, int* lrwork, int* iwork, int* liwork,
                int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldz = LDZ, il, iu, m, info, lwork, lrwork, liwork;
        double abstol, vl, vu;
        int iwkopt;
        int* iwork;
        double rwkopt;
        double* rwork;
        dcomplex wkopt;
```

```
        dcomplex* work;
        /* Local arrays */
        int isuppz[N];
        double w[N];
        dcomplex z[LDZ*N];
        dcomplex a[LDA*N] = {
           {-2.16,  0.00}, {-0.16,  4.86}, {-7.23,  9.38}, {-0.04, -6.86},
           { 0.00,  0.00}, { 7.45,  0.00}, { 4.39, -6.29}, {-8.11,  4.41},
           { 0.00,  0.00}, { 0.00,  0.00}, {-9.03,  0.00}, {-6.89,  7.66},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 7.76,  0.00}
        };
        /* Executable statements */
        printf( " ZHEEVR Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = -5.0;
        vu = 5.0;
        /* Query and allocate the optimal workspace */
        lwork = -1;
        lrwork = -1;
        liwork = -1;
        zheevr( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, &wkopt, &lwork, &rwkopt, &lrwork,
                        &iwkopt, &liwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        lrwork = (int)rwkopt;
        rwork = (double*)malloc( lrwork*sizeof(double) );
        liwork = iwkopt;
        iwork = (int*)malloc( liwork*sizeof(int) );
        /* Solve eigenproblem */
        zheevr( "Vectors", "Values", "Lower", &n, a, &lda, &vl, &vu, &il, &iu,
                        &abstol, &m, w, z, &ldz, isuppz, work, &lwork, rwork, &lrwork,
                        iwork, &liwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        /* Free workspace */
        free( (void*)iwork );
        free( (void*)rwork );
        free( (void*)work );
        exit( 0 );
} /* End of ZHEEVR Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zheevr Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheevr Example.
   =======================

   Program computes eigenvalues specified by a selected range of values
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
```

```
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_zheevr (row-major, high-level) Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57

 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        double w[N];
        MKL_Complex16 z[LDZ*N];
        MKL_Complex16 a[LDA*N] = {
           {-2.16,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-0.16,  4.86}, { 7.45,  0.00}, { 0.00,  0.00}, { 0.00,  0.00},
           {-7.23,  9.38}, { 4.39, -6.29}, {-9.03,  0.00}, { 0.00,  0.00},
           {-0.04, -6.86}, {-8.11,  4.41}, {-6.89,  7.66}, { 7.76,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevr (row-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = -5.0;
        vu = 5.0;
        /* Solve eigenproblem */
        info = LAPACKE_zheevr( LAPACK_ROW_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
```

```
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_zheevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_zheevr Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zheevr Example.
   ======================

   Program computes eigenvalues specified by a selected range of values
```

```
   and corresponding eigenvectors of a complex Hermitian matrix A using the
   Relatively Robust Representations, where A is:

   ( -2.16,  0.00) ( -0.16, -4.86) ( -7.23, -9.38) ( -0.04,  6.86)
   ( -0.16,  4.86) (  7.45,  0.00) (  4.39,  6.29) ( -8.11, -4.41)
   ( -7.23,  9.38) (  4.39, -6.29) ( -9.03,  0.00) ( -6.89, -7.66)
   ( -0.04, -6.86) ( -8.11,  4.41) ( -6.89,  7.66) (  7.76,  0.00)

   Description.
   ============

   The routine computes selected eigenvalues and, optionally, eigenvectors of
   an n-by-n complex Hermitian matrix A. The eigenvector v(j) of A satisfies

   A*v(j) = lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The computed eigenvectors are
   orthonormal.
   Eigenvalues and eigenvectors can be selected by specifying either a range
   of values or a range of indices for the desired eigenvalues.

   Example Program Results.
   ========================

 LAPACKE_zheevr (column-major, high-level) Example Program Results

 The total number of eigenvalues found: 2

 Selected eigenvalues
  -4.18   3.57

 Selected eigenvectors (stored columnwise)
 (  0.68,  0.00) (  0.38,  0.00)
 (  0.03,  0.18) (  0.54, -0.57)
 ( -0.03,  0.21) ( -0.40,  0.04)
 (  0.20,  0.64) ( -0.14, -0.26)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDZ N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldz = LDZ, il, iu, m, info;
        double abstol, vl, vu;
        /* Local arrays */
        MKL_INT isuppz[N];
        double w[N];
```

```
        MKL_Complex16 z[LDZ*N];
        MKL_Complex16 a[LDA*N] = {
           {-2.16,  0.00}, {-0.16,  4.86}, {-7.23,  9.38}, {-0.04, -6.86},
           { 0.00,  0.00}, { 7.45,  0.00}, { 4.39, -6.29}, {-8.11,  4.41},
           { 0.00,  0.00}, { 0.00,  0.00}, {-9.03,  0.00}, {-6.89,  7.66},
           { 0.00,  0.00}, { 0.00,  0.00}, { 0.00,  0.00}, { 7.76,  0.00}
        };
        /* Executable statements */
        printf( "LAPACKE_zheevr (column-major, high-level) Example Program Results\n" );
        /* Negative abstol means using the default value */
        abstol = -1.0;
        /* Set VL, VU to compute eigenvalues in half-open (VL,VU] interval */
        vl = -5.0;
        vu = 5.0;
        /* Solve eigenproblem */
        info = LAPACKE_zheevr( LAPACK_COL_MAJOR, 'V', 'V', 'L', n, a, lda,
                        vl, vu, il, iu, abstol, &m, w, z, ldz, isuppz );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print the number of eigenvalues found */
        printf( "\n The total number of eigenvalues found:%2i\n", m );
        /* Print eigenvalues */
        print_rmatrix( "Selected eigenvalues", 1, m, w, 1 );
        /* Print eigenvectors */
        print_matrix( "Selected eigenvectors (stored columnwise)", n, m, z, ldz );
        exit( 0 );
} /* End of LAPACKE_zheevr Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

Nonsymmetric Eigenproblems

Routine	Description	Examples
?geev	Computes the eigenvalues and, optionally, the left and/or right eigenvectors of a general matrix.	cgeev dgeev sgeev zgeev

?geev function

Computes the eigenvalues and left and right eigenvectors of a general matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork, info)
call dgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork, info)
call cgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork, info)
call zgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork, info)
```

Calling from C:

```
sgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork, info);
dgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork, info);
cgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork, info);
zgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork, info);
```

C Interface

```
lapack_int LAPACKE_sgeev(int matrix_layout, char jobvl, char jobvr, lapack_int n, float* a,
lapack_int lda, float* wr, float* wi, float* vl, lapack_int ldvl, float* vr, lapack_int
ldvr);
lapack_int LAPACKE_dgeev(int matrix_layout, char jobvl, char jobvr, lapack_int n, double*
a, lapack_int lda, double* wr, double* wi, double* vl, lapack_int ldvl, double* vr,
lapack_int ldvr);
lapack_int LAPACKE_cgeev(int matrix_layout, char jobvl, char jobvr, lapack_int n,
lapack_complex_float* a, lapack_int lda, lapack_complex_float* w, lapack_complex_float* vl,
lapack_int ldvl, lapack_complex_float* vr, lapack_int ldvr);
lapack_int LAPACKE_zgeev(int matrix_layout, char jobvl, char jobvr, lapack_int n,
lapack_complex_double* a, lapack_int lda, lapack_complex_double* w, lapack_complex_double*
vl, lapack_int ldvl, lapack_complex_double* vr, lapack_int ldvr);
```

See Also

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CGEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the eigenvalues and, optionally, the left and/or right eigenvectors of a square complex general matrix *A*.

The right eigenvector $v(j)$ of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

$lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of *A* satisfies the following formula:

```
u(j)^H*A = lambda(j)*u(j)^H
```

where

$u(j)^H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized so that their Euclidean norm equals one and the largest component is real.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
```

```
*  =====================================================================
*
*  CGEEV Example.
*  ==============
*
*  Program computes the eigenvalues and left and right eigenvectors of a general
*  rectangular matrix A:
*
*  ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
*  ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
*  ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
*  (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)
*
*  Description.
*  ============
*
*  The routine computes for an n-by-n complex nonsymmetric matrix A, the
*  eigenvalues and, optionally, the left and/or right eigenvectors. The right
*  eigenvector v(j) of A satisfies
*
*  A*v(j)= lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies
*
*  u(j)H*A = lambda(j)*u(j)H
*
*  where u(j)H denotes the conjugate transpose of u(j). The computed
*  eigenvectors are normalized to have Euclidean norm equal to 1 and
*  largest component real.
*
*  Example Program Results.
*  ========================
*
* CGEEV Example Program Results
*
* Eigenvalues
* ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)
*
* Left eigenvectors
* (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
* (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
* (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
* ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)
*
* Right eigenvectors
* (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
* (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
* (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
* ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDVL, LDVR
      PARAMETER        ( LDA = N, LDVL = N, LDVR = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
```

```
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least 2*N
      REAL             RWORK( 2*N )
      COMPLEX          A( LDA, N ), VL( LDVL, N ), VR( LDVR, N ),
     $                 W( N ), WORK( LWMAX )
      DATA             A/
     $ (-3.84, 2.25),(-0.66, 0.83),(-3.99,-4.73),( 7.74, 4.18),
     $ (-8.94,-4.75),(-4.40,-3.82),(-5.88,-6.60),( 3.66,-7.53),
     $ ( 8.95,-6.53),(-3.50,-4.26),(-3.36,-0.40),( 2.58, 3.60),
     $ (-9.87, 4.82),(-3.15, 7.36),(-0.75, 5.23),( 4.59, 5.41)
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         CGEEV
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CGEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CGEEV( 'Vectors', 'Vectors', N, A, LDA, W, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, RWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL CGEEV( 'Vectors', 'Vectors', N, A, LDA, W, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, RWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print left eigenvectors.
*
      CALL PRINT_MATRIX( 'Left eigenvectors', N, N, VL, LDVL )
*
*     Print right eigenvectors.
*
      CALL PRINT_MATRIX( 'Right eigenvectors', N, N, VR, LDVR )
      STOP
      END
```

```
*
*     End of CGEEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

CGEEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CGEEV Example.
   ==============

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

   Description.
   ============
```

```
   The routine computes for an n-by-n complex nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 CGEEV Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGEEV prototype */
extern void cgeev( char* jobvl, char* jobvr, int* n, fcomplex* a,
                int* lda, fcomplex* w, fcomplex* vl, int* ldvl, fcomplex* vr, int* ldvr,
                fcomplex* work, int* lwork, float* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info, lwork;
```

```
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least 2*n */
        float rwork[2*N];
        fcomplex w[N], vl[LDVL*N], vr[LDVR*N];
        fcomplex a[LDA*N] = {
           {-3.84f,  2.25f}, {-0.66f,  0.83f}, {-3.99f, -4.73f}, { 7.74f,  4.18f},
           {-8.94f, -4.75f}, {-4.40f, -3.82f}, {-5.88f, -6.60f}, { 3.66f, -7.53f},
           { 8.95f, -6.53f}, {-3.50f, -4.26f}, {-3.36f, -0.40f}, { 2.58f,  3.60f},
           {-9.87f,  4.82f}, {-3.15f,  7.36f}, {-0.75f,  5.23f}, { 4.59f,  5.41f}
        };
        /* Executable statements */
        printf( " CGEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cgeev( "Vectors", "Vectors", &n, a, &lda, w, vl, &ldvl, vr, &ldvr,
         &wkopt, &lwork, rwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Solve eigenproblem */
        cgeev( "Vectors", "Vectors", &n, a, &lda, w, vl, &ldvl, vr, &ldvr,
         work, &lwork, rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CGEEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_cgeev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
```

```
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

   Description.
   ============

   The routine computes for an n-by-n complex nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 LAPACKE_cgeev (row-major, high-level) Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
```

```
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        MKL_Complex8 w[N], vl[LDVL*N], vr[LDVR*N];
        MKL_Complex8 a[LDA*N] = {
           {-3.84f,  2.25f}, {-8.94f, -4.75f}, { 8.95f, -6.53f}, {-9.87f,  4.82f},
           {-0.66f,  0.83f}, {-4.40f, -3.82f}, {-3.50f, -4.26f}, {-3.15f,  7.36f},
           {-3.99f, -4.73f}, {-5.88f, -6.60f}, {-3.36f, -0.40f}, {-0.75f,  5.23f},
           { 7.74f,  4.18f}, { 3.66f, -7.53f}, { 2.58f,  3.60f}, { 4.59f,  5.41f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgeev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cgeev( LAPACK_ROW_MAJOR, 'V', 'V', n, a, lda, w, vl,
         ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_cgeev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

LAPACKE_cgeev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

   Description.
   ============

   The routine computes for an n-by-n complex nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 LAPACKE_cgeev (column-major, high-level) Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
```

```
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        MKL_Complex8 w[N], vl[LDVL*N], vr[LDVR*N];
        MKL_Complex8 a[LDA*N] = {
           {-3.84f,  2.25f}, {-0.66f,  0.83f}, {-3.99f, -4.73f}, { 7.74f,  4.18f},
           {-8.94f, -4.75f}, {-4.40f, -3.82f}, {-5.88f, -6.60f}, { 3.66f, -7.53f},
           { 8.95f, -6.53f}, {-3.50f, -4.26f}, {-3.36f, -0.40f}, { 2.58f,  3.60f},
           {-9.87f,  4.82f}, {-3.15f,  7.36f}, {-0.75f,  5.23f}, { 4.59f,  5.41f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgeev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_cgeev( LAPACK_COL_MAJOR, 'V', 'V', n, a, lda, w, vl,
         ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_cgeev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

DGEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes a square real general matrix *A*, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector $v(j)$ of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

$lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of *A* satisfies the following formula:

$u(j)^H*A = lambda(j)*u(j)^H$

where

$u(j)^H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized so that their Euclidean norm equals one and the largest component is real.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
```

```
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DGEEV Example.
*  ==============
*
*  Program computes the eigenvalues and left and right eigenvectors of a general
*  rectangular matrix A:
*
*   -1.01   0.86  -4.60   3.31  -4.81
*    3.98   0.53  -7.04   5.29   3.55
*    3.30   8.26  -3.89   8.20  -1.51
*    4.43   4.96  -7.66  -7.33   6.18
*    7.31  -6.43  -6.16   2.47   5.58
*
*  Description.
*  ============
*
*  The routine computes for an n-by-n real nonsymmetric matrix A, the
*  eigenvalues and, optionally, the left and/or right eigenvectors. The right
*  eigenvector v(j) of A satisfies
*
*  A*v(j)= lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies
*
*  u(j)H*A = lambda(j)*u(j)H
*
*  where u(j)H denotes the conjugate transpose of u(j). The computed
*  eigenvectors are normalized to have Euclidean norm equal to 1 and
*  largest component real.
*
*  Example Program Results.
*  ========================
*
* DGEEV Example Program Results
*
* Eigenvalues
* (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46
*
* Left eigenvectors
* (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
* (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
* ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
* (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
* ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18
*
* Right eigenvectors
* (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
* (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
* (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
* (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
```

```
* (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA, LDVL, LDVR
      PARAMETER        ( LDA = N, LDVL = N, LDVR = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      DOUBLE PRECISION A( LDA, N ), VL( LDVL, N ), VR( LDVR, N ),
     $                 WR( N ), WI( N ), WORK( LWMAX )
      DATA             A/
     $ -1.01, 3.98, 3.30, 4.43, 7.31,
     $  0.86, 0.53, 8.26, 4.96,-6.43,
     $ -4.60,-7.04,-3.89,-7.66,-6.16,
     $  3.31, 5.29, 8.20,-7.33, 2.47,
     $ -4.81, 3.55,-1.51, 6.18, 5.58
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DGEEV
      EXTERNAL         PRINT_EIGENVALUES, PRINT_EIGENVECTORS
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DGEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DGEEV( 'Vectors', 'Vectors', N, A, LDA, WR, WI, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL DGEEV( 'Vectors', 'Vectors', N, A, LDA, WR, WI, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_EIGENVALUES( 'Eigenvalues', N, WR, WI )
*
```

```
*     Print left eigenvectors.
*
      CALL PRINT_EIGENVECTORS( 'Left eigenvectors', N, WI, VL, LDVL )
*
*     Print right eigenvectors.
*
      CALL PRINT_EIGENVECTORS( 'Right eigenvectors', N, WI, VR, LDVR )
      STOP
      END
*
*     End of DGEEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing eigenvalues.
*
      SUBROUTINE PRINT_EIGENVALUES( DESC, N, WR, WI )
      CHARACTER*(*)    DESC
      INTEGER          N
      DOUBLE PRECISION WR( * ), WI( * )
*
      DOUBLE PRECISION ZERO
      PARAMETER        ( ZERO = 0.0 )
      INTEGER          J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO J = 1, N
         IF( WI( J ).EQ.ZERO ) THEN
            WRITE(*,9998,ADVANCE='NO') WR( J )
         ELSE
            WRITE(*,9999,ADVANCE='NO') WR( J ), WI( J )
         END IF
      END DO
      WRITE(*,*)
*
 9998 FORMAT( 11(:,1X,F6.2) )
 9999 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing eigenvectors.
*
      SUBROUTINE PRINT_EIGENVECTORS( DESC, N, WI, V, LDV )
      CHARACTER*(*)    DESC
      INTEGER          N, LDV
      DOUBLE PRECISION WI( * ), V( LDV, * )
*
      DOUBLE PRECISION ZERO
      PARAMETER        ( ZERO = 0.0 )
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, N
         J = 1
         DO WHILE( J.LE.N )
            IF( WI( J ).EQ.ZERO ) THEN
```

```
               WRITE(*,9998,ADVANCE='NO') V( I, J )
               J = J + 1
            ELSE
               WRITE(*,9999,ADVANCE='NO') V( I, J ), V( I, J+1 )
               WRITE(*,9999,ADVANCE='NO') V( I, J ), -V( I, J+1 )
               J = J + 2
            END IF
         END DO
         WRITE(*,*)
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
 9999 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

DGEEV Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   DGEEV Example.
   ==============

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies
```

```
  u(j)H*A = lambda(j)*u(j)H

  where u(j)H denotes the conjugate transpose of u(j). The computed
  eigenvectors are normalized to have Euclidean norm equal to 1 and
  largest component real.

  Example Program Results.
  ========================

 DGEEV Example Program Results

 Eigenvalues
 (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

 Left eigenvectors
 (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>

/* DGEEV prototype */
extern void dgeev( char* jobvl, char* jobvr, int* n, double* a,
                int* lda, double* wr, double* wi, double* vl, int* ldvl,
                double* vr, int* ldvr, double* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, int n, double* wr, double* wi );
extern void print_eigenvectors( char* desc, int n, double* wi, double* v,
      int ldv );

/* Parameters */
#define N 5
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info, lwork;
        double wkopt;
        double* work;
        /* Local arrays */
        double wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        double a[LDA*N] = {
           -1.01,  3.98,  3.30,  4.43,  7.31,
            0.86,  0.53,  8.26,  4.96, -6.43,
```

```
           -4.60, -7.04, -3.89, -7.66, -6.16,
            3.31,  5.29,  8.20, -7.33,  2.47,
           -4.81,  3.55, -1.51,  6.18,  5.58
        };
        /* Executable statements */
        printf( " DGEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dgeev( "Vectors", "Vectors", &n, a, &lda, wr, wi, vl, &ldvl, vr, &ldvr,
         &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Solve eigenproblem */
        dgeev( "Vectors", "Vectors", &n, a, &lda, wr, wi, vl, &ldvl, vr, &ldvr,
         work, &lwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DGEEV Example */

/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, int n, double* wr, double* wi ) {
        int j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (double)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}

/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, int n, double* wi, double* v, int ldv ) {
        int i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (double)0.0 ) {
            printf( " %6.2f", v[i+j*ldv] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], v[i+(j+1)*ldv] );
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], -v[i+(j+1)*ldv] );
```

```
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

LAPACKE_dgeev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================
```

```
 LAPACKE_dgeev (row-major, high-level) Example Program Results

 Eigenvalues
 (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

 Left eigenvectors
 (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, MKL_INT n, double* wr, double* wi );
extern void print_eigenvectors( char* desc, MKL_INT n, double* wi, double* v,
      MKL_INT ldv );

/* Parameters */
#define N 5
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        double wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        double a[LDA*N] = {
           -1.01,  0.86, -4.60,  3.31, -4.81,
            3.98,  0.53, -7.04,  5.29,  3.55,
            3.30,  8.26, -3.89,  8.20, -1.51,
            4.43,  4.96, -7.66, -7.33,  6.18,
            7.31, -6.43, -6.16,  2.47,  5.58
        };
        /* Executable statements */
        printf( "LAPACKE_dgeev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dgeev( LAPACK_ROW_MAJOR, 'V', 'V', n, a, lda, wr, wi,
                        vl, ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
```

```
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_dgeev Example */

/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, MKL_INT n, double* wr, double* wi ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (double)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}

/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, MKL_INT n, double* wi, double* v, MKL_INT ldv ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (double)0.0 ) {
            printf( " %6.2f", v[i*ldv+j] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i*ldv+j], v[i*ldv+(j+1)] );
            printf( " (%6.2f,%6.2f)", v[i*ldv+j], -v[i*ldv+(j+1)] );
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

LAPACKE_dgeev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
```

```
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 LAPACKE_dgeev (column-major, high-level) Example Program Results

 Eigenvalues
 (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

 Left eigenvectors
 (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>
```

```
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, MKL_INT n, double* wr, double* wi );
extern void print_eigenvectors( char* desc, MKL_INT n, double* wi, double* v,
      MKL_INT ldv );

/* Parameters */
#define N 5
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        double wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        double a[LDA*N] = {
           -1.01,  3.98,  3.30,  4.43,  7.31,
            0.86,  0.53,  8.26,  4.96, -6.43,
           -4.60, -7.04, -3.89, -7.66, -6.16,
            3.31,  5.29,  8.20, -7.33,  2.47,
           -4.81,  3.55, -1.51,  6.18,  5.58
        };
        /* Executable statements */
        printf( "LAPACKE_dgeev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_dgeev( LAPACK_COL_MAJOR, 'V', 'V', n, a, lda, wr, wi,
                        vl, ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_dgeev Example */

/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, MKL_INT n, double* wr, double* wi ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (double)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}
```

```
/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, MKL_INT n, double* wi, double* v, MKL_INT ldv ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (double)0.0 ) {
            printf( " %6.2f", v[i+j*ldv] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], v[i+(j+1)*ldv] );
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], -v[i+(j+1)*ldv] );
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

SGEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the eigenvalues and, optionally, the left and/or right eigenvectors of a square real general matrix *A*.

The right eigenvector $v(j)$ of *A* satisfies the following formula:

```
A*v(j) = lambda(j)*v(j)
```

where

$lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of *A* satisfies the following formula:

```
u(j)^H*A = lambda(j)*u(j)^H
```

where

$u(j)^H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized so that their Euclidean norm equals one and the largest component is real.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C
?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SGEEV Example.
*  ==============
*
*  Program computes the eigenvalues and left and right eigenvectors of a general
*  rectangular matrix A:
*
*   -1.01   0.86  -4.60   3.31  -4.81
*    3.98   0.53  -7.04   5.29   3.55
*    3.30   8.26  -3.89   8.20  -1.51
*    4.43   4.96  -7.66  -7.33   6.18
*    7.31  -6.43  -6.16   2.47   5.58
*
*  Description.
*  ============
*
*  The routine computes for an n-by-n real nonsymmetric matrix A, the
*  eigenvalues and, optionally, the left and/or right eigenvectors. The right
*  eigenvector v(j) of A satisfies
*
*  A*v(j)= lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies
*
*  u(j)H*A = lambda(j)*u(j)H
*
*  where u(j)H denotes the conjugate transpose of u(j). The computed
*  eigenvectors are normalized to have Euclidean norm equal to 1 and
*  largest component real.
*
*  Example Program Results.
*  ========================
*
* SGEEV Example Program Results
*
* Eigenvalues
```

```
* (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46
*
* Left eigenvectors
* (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
* (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
* ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
* (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
* ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18
*
* Right eigenvectors
* (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
* (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
* (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
* (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
* (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 5 )
      INTEGER          LDA, LDVL, LDVR
      PARAMETER        ( LDA = N, LDVL = N, LDVR = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      REAL             A( LDA, N ), VL( LDVL, N ), VR( LDVR, N ),
     $                 WR( N ), WI( N ), WORK( LWMAX )
      DATA             A/
     $ -1.01, 3.98, 3.30, 4.43, 7.31,
     $  0.86, 0.53, 8.26, 4.96,-6.43,
     $ -4.60,-7.04,-3.89,-7.66,-6.16,
     $  3.31, 5.29, 8.20,-7.33, 2.47,
     $ -4.81, 3.55,-1.51, 6.18, 5.58
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SGEEV
      EXTERNAL         PRINT_EIGENVALUES, PRINT_EIGENVECTORS
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SGEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL SGEEV( 'Vectors', 'Vectors', N, A, LDA, WR, WI, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
```

```
      CALL SGEEV( 'Vectors', 'Vectors', N, A, LDA, WR, WI, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_EIGENVALUES( 'Eigenvalues', N, WR, WI )
*
*     Print left eigenvectors.
*
      CALL PRINT_EIGENVECTORS( 'Left eigenvectors', N, WI, VL, LDVL )
*
*     Print right eigenvectors.
*
      CALL PRINT_EIGENVECTORS( 'Right eigenvectors', N, WI, VR, LDVR )
      STOP
      END
*
*     End of SGEEV Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing eigenvalues.
*
      SUBROUTINE PRINT_EIGENVALUES( DESC, N, WR, WI )
      CHARACTER*(*)    DESC
      INTEGER          N
      REAL             WR( * ), WI( * )
*
      REAL             ZERO
      PARAMETER        ( ZERO = 0.0 )
      INTEGER          J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO J = 1, N
         IF( WI( J ).EQ.ZERO ) THEN
            WRITE(*,9998,ADVANCE='NO') WR( J )
         ELSE
            WRITE(*,9999,ADVANCE='NO') WR( J ), WI( J )
         END IF
      END DO
      WRITE(*,*)
*
 9998 FORMAT( 11(:,1X,F6.2) )
 9999 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing eigenvectors.
*
      SUBROUTINE PRINT_EIGENVECTORS( DESC, N, WI, V, LDV )
```

```
      CHARACTER*(*)    DESC
      INTEGER          N, LDV
      REAL             WI( * ), V( LDV, * )
*
      REAL             ZERO
      PARAMETER        ( ZERO = 0.0 )
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, N
         J = 1
         DO WHILE( J.LE.N )
            IF( WI( J ).EQ.ZERO ) THEN
               WRITE(*,9998,ADVANCE='NO') V( I, J )
               J = J + 1
            ELSE
               WRITE(*,9999,ADVANCE='NO') V( I, J ), V( I, J+1 )
               WRITE(*,9999,ADVANCE='NO') V( I, J ), -V( I, J+1 )
               J = J + 2
            END IF
         END DO
         WRITE(*,*)
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
 9999 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

SGEEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   SGEEV Example.
   ==============

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
```

```
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 SGEEV Example Program Results

 Eigenvalues
 (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

 Left eigenvectors
 (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>

/* SGEEV prototype */
extern void sgeev( char* jobvl, char* jobvr, int* n, float* a,
                int* lda, float* wr, float* wi, float* vl, int* ldvl,
                float* vr, int* ldvr, float* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, int n, float* wr, float* wi );
extern void print_eigenvectors( char* desc, int n, float* wi, float* v,
      int ldv );

/* Parameters */
#define N 5
#define LDA N
```

```
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
        float wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        float a[LDA*N] = {
           -1.01f,  3.98f,  3.30f,  4.43f,  7.31f,
            0.86f,  0.53f,  8.26f,  4.96f, -6.43f,
           -4.60f, -7.04f, -3.89f, -7.66f, -6.16f,
            3.31f,  5.29f,  8.20f, -7.33f,  2.47f,
           -4.81f,  3.55f, -1.51f,  6.18f,  5.58f
        };
        /* Executable statements */
        printf( " SGEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        sgeev( "Vectors", "Vectors", &n, a, &lda, wr, wi, vl, &ldvl, vr, &ldvr,
         &wkopt, &lwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Solve eigenproblem */
        sgeev( "Vectors", "Vectors", &n, a, &lda, wr, wi, vl, &ldvl, vr, &ldvr,
         work, &lwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SGEEV Example */

/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, int n, float* wr, float* wi ) {
        int j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (float)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}
```

```
/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, int n, float* wi, float* v, int ldv ) {
        int i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (float)0.0 ) {
            printf( " %6.2f", v[i+j*ldv] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], v[i+(j+1)*ldv] );
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], -v[i+(j+1)*ldv] );
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

LAPACKE_sgeev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
```

```
  eigenvector v(j) of A satisfies

  A*v(j)= lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

  u(j)H*A = lambda(j)*u(j)H

  where u(j)H denotes the conjugate transpose of u(j). The computed
  eigenvectors are normalized to have Euclidean norm equal to 1 and
  largest component real.

  Example Program Results.
  ========================

 LAPACKE_sgeev (row-major, high-level) Example Program Results

 Eigenvalues
 (  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

 Left eigenvectors
 (  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, MKL_INT n, float* wr, float* wi );
extern void print_eigenvectors( char* desc, MKL_INT n, float* wi, float* v,
      MKL_INT ldv );

/* Parameters */
#define N 5
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        float wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        float a[LDA*N] = {
           -1.01f,  0.86f, -4.60f,  3.31f, -4.81f,
            3.98f,  0.53f, -7.04f,  5.29f,  3.55f,
```

```
            3.30f,  8.26f, -3.89f,  8.20f, -1.51f,
            4.43f,  4.96f, -7.66f, -7.33f,  6.18f,
            7.31f, -6.43f, -6.16f,  2.47f,  5.58f
        };
        /* Executable statements */
        printf( "LAPACKE_sgeev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_sgeev( LAPACK_ROW_MAJOR, 'V', 'V', n, a, lda, wr, wi,
                        vl, ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_sgeev Example */

/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, MKL_INT n, float* wr, float* wi ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (float)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}

/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, MKL_INT n, float* wi, float* v, MKL_INT ldv ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (float)0.0 ) {
            printf( " %6.2f", v[i*ldv+j] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i*ldv+j], v[i*ldv+(j+1)] );
            printf( " (%6.2f,%6.2f)", v[i*ldv+j], -v[i*ldv+(j+1)] );
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

LAPACKE_sgeev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

    -1.01   0.86  -4.60   3.31  -4.81
     3.98   0.53  -7.04   5.29   3.55
     3.30   8.26  -3.89   8.20  -1.51
     4.43   4.96  -7.66  -7.33   6.18
     7.31  -6.43  -6.16   2.47   5.58

   Description.
   ============

   The routine computes for an n-by-n real nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

LAPACKE_sgeev (column-major, high-level) Example Program Results

Eigenvalues
(  2.86, 10.76) (  2.86,-10.76) ( -0.69,  4.70) ( -0.69, -4.70) -10.46

Left eigenvectors
(  0.04,  0.29) (  0.04, -0.29) ( -0.13, -0.33) ( -0.13,  0.33)   0.04
```

```
 (  0.62,  0.00) (  0.62,  0.00) (  0.69,  0.00) (  0.69,  0.00)   0.56
 ( -0.04, -0.58) ( -0.04,  0.58) ( -0.39, -0.07) ( -0.39,  0.07)  -0.13
 (  0.28,  0.01) (  0.28, -0.01) ( -0.02, -0.19) ( -0.02,  0.19)  -0.80
 ( -0.04,  0.34) ( -0.04, -0.34) ( -0.40,  0.22) ( -0.40, -0.22)   0.18

 Right eigenvectors
 (  0.11,  0.17) (  0.11, -0.17) (  0.73,  0.00) (  0.73,  0.00)   0.46
 (  0.41, -0.26) (  0.41,  0.26) ( -0.03, -0.02) ( -0.03,  0.02)   0.34
 (  0.10, -0.51) (  0.10,  0.51) (  0.19, -0.29) (  0.19,  0.29)   0.31
 (  0.40, -0.09) (  0.40,  0.09) ( -0.08, -0.08) ( -0.08,  0.08)  -0.74
 (  0.54,  0.00) (  0.54,  0.00) ( -0.29, -0.49) ( -0.29,  0.49)   0.16
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_eigenvalues( char* desc, MKL_INT n, float* wr, float* wi );
extern void print_eigenvectors( char* desc, MKL_INT n, float* wi, float* v,
      MKL_INT ldv );

/* Parameters */
#define N 5
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        float wr[N], wi[N], vl[LDVL*N], vr[LDVR*N];
        float a[LDA*N] = {
           -1.01f,  3.98f,  3.30f,  4.43f,  7.31f,
            0.86f,  0.53f,  8.26f,  4.96f, -6.43f,
           -4.60f, -7.04f, -3.89f, -7.66f, -6.16f,
            3.31f,  5.29f,  8.20f, -7.33f,  2.47f,
           -4.81f,  3.55f, -1.51f,  6.18f,  5.58f
        };
        /* Executable statements */
        printf( "LAPACKE_sgeev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_sgeev( LAPACK_COL_MAJOR, 'V', 'V', n, a, lda, wr, wi,
                        vl, ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_eigenvalues( "Eigenvalues", n, wr, wi );
        /* Print left eigenvectors */
        print_eigenvectors( "Left eigenvectors", n, wi, vl, ldvl );
        /* Print right eigenvectors */
        print_eigenvectors( "Right eigenvectors", n, wi, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_sgeev Example */
```

```
/* Auxiliary routine: printing eigenvalues */
void print_eigenvalues( char* desc, MKL_INT n, float* wr, float* wi ) {
        MKL_INT j;
        printf( "\n %s\n", desc );
   for( j = 0; j < n; j++ ) {
      if( wi[j] == (float)0.0 ) {
         printf( " %6.2f", wr[j] );
      } else {
         printf( " (%6.2f,%6.2f)", wr[j], wi[j] );
      }
   }
   printf( "\n" );
}

/* Auxiliary routine: printing eigenvectors */
void print_eigenvectors( char* desc, MKL_INT n, float* wi, float* v, MKL_INT ldv ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
   for( i = 0; i < n; i++ ) {
      j = 0;
      while( j < n ) {
         if( wi[j] == (float)0.0 ) {
            printf( " %6.2f", v[i+j*ldv] );
            j++;
         } else {
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], v[i+(j+1)*ldv] );
            printf( " (%6.2f,%6.2f)", v[i+j*ldv], -v[i+(j+1)*ldv] );
            j += 2;
         }
      }
      printf( "\n" );
   }
}
```

ZGEEV Example

NOTE
For a detailed description and reference information on this function, please visit:

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the eigenvalues and, optionally, the left and/or right eigenvectors of a square complex general matrix *A*.

The right eigenvector $v(j)$ of *A* satisfies the following formula:

$$A*v(j) = lambda(j)*v(j)$$

where

$lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of *A* satisfies the following formula:

$$u(j)^H*A = lambda(j)*u(j)^H$$

where

$u(j)^H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized so that their Euclidean norm equals one and the largest component is real.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - C

?geev function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGEEV Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  ZGEEV Example.
*  ==============
*
*  Program computes the eigenvalues and left and right eigenvectors of a general
*  rectangular matrix A:
*
*  ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
*  ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
*  ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
*  (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)
*
*  Description.
*  ============
*
*  The routine computes for an n-by-n complex nonsymmetric matrix A, the
*  eigenvalues and, optionally, the left and/or right eigenvectors. The right
*  eigenvector v(j) of A satisfies
*
*  A*v(j)= lambda(j)*v(j)
*
*  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies
```

```
*
*  u(j)H*A = lambda(j)*u(j)H
*
*  where u(j)H denotes the conjugate transpose of u(j). The computed
*  eigenvectors are normalized to have Euclidean norm equal to 1 and
*  largest component real.
*
*  Example Program Results.
*  ========================
*
* ZGEEV Example Program Results
*
* Eigenvalues
* ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)
*
* Left eigenvectors
* (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
* (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
* (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
* ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)
*
* Right eigenvectors
* (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
* (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
* (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
* ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*  ================================================================================
*
*     .. Parameters ..
      INTEGER          N
      PARAMETER        ( N = 4 )
      INTEGER          LDA, LDVL, LDVR
      PARAMETER        ( LDA = N, LDVL = N, LDVR = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least 2*N
      DOUBLE PRECISION RWORK( 2*N )
      COMPLEX*16       A( LDA, N ), VL( LDVL, N ), VR( LDVR, N ),
     $                 W( N ), WORK( LWMAX )
      DATA             A/
     $ (-3.84, 2.25),(-0.66, 0.83),(-3.99,-4.73),( 7.74, 4.18),
     $ (-8.94,-4.75),(-4.40,-3.82),(-5.88,-6.60),( 3.66,-7.53),
     $ ( 8.95,-6.53),(-3.50,-4.26),(-3.36,-0.40),( 2.58, 3.60),
     $ (-9.87, 4.82),(-3.15, 7.36),(-0.75, 5.23),( 4.59, 5.41)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZGEEV
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
```

```
*     .. Executable Statements ..
      WRITE(*,*)'ZGEEV Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZGEEV( 'Vectors', 'Vectors', N, A, LDA, W, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, RWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Solve eigenproblem.
*
      CALL ZGEEV( 'Vectors', 'Vectors', N, A, LDA, W, VL, LDVL,
     $            VR, LDVR, WORK, LWORK, RWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm failed to compute eigenvalues.'
         STOP
      END IF
*
*     Print eigenvalues.
*
      CALL PRINT_MATRIX( 'Eigenvalues', 1, N, W, 1 )
*
*     Print left eigenvectors.
*
      CALL PRINT_MATRIX( 'Left eigenvectors', N, N, VL, LDVL )
*
*     Print right eigenvectors.
*
      CALL PRINT_MATRIX( 'Right eigenvectors', N, N, VR, LDVR )
      STOP
      END
*
*     End of ZGEEV Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
```

ZGEEV Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZGEEV Example.
   ==============

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

   Description.
   ============

   The routine computes for an n-by-n complex nonsymmetric matrix A, the
   eigenvalues and, optionally, the left and/or right eigenvectors. The right
   eigenvector v(j) of A satisfies

   A*v(j)= lambda(j)*v(j)

   where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

   u(j)H*A = lambda(j)*u(j)H

   where u(j)H denotes the conjugate transpose of u(j). The computed
   eigenvectors are normalized to have Euclidean norm equal to 1 and
   largest component real.

   Example Program Results.
   ========================

 ZGEEV Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
```

```
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGEEV prototype */
extern void zgeev( char* jobvl, char* jobvr, int* n, dcomplex* a,
                int* lda, dcomplex* w, dcomplex* vl, int* ldvl, dcomplex* vr, int* ldvr,
                dcomplex* work, int* lwork, double* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        int n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least 2*n */
        double rwork[2*N];
        dcomplex w[N], vl[LDVL*N], vr[LDVR*N];
        dcomplex a[LDA*N] = {
           {-3.84,  2.25}, {-0.66,  0.83}, {-3.99, -4.73}, { 7.74,  4.18},
           {-8.94, -4.75}, {-4.40, -3.82}, {-5.88, -6.60}, { 3.66, -7.53},
           { 8.95, -6.53}, {-3.50, -4.26}, {-3.36, -0.40}, { 2.58,  3.60},
           {-9.87,  4.82}, {-3.15,  7.36}, {-0.75,  5.23}, { 4.59,  5.41}
        };
        /* Executable statements */
        printf( " ZGEEV Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zgeev( "Vectors", "Vectors", &n, a, &lda, w, vl, &ldvl, vr, &ldvr,
         &wkopt, &lwork, rwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Solve eigenproblem */
        zgeev( "Vectors", "Vectors", &n, a, &lda, w, vl, &ldvl, vr, &ldvr,
         work, &lwork, rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
```

```
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZGEEV Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}
```

LAPACKE_zgeev Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:

   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

   Description.
   ============
```

```
  The routine computes for an n-by-n complex nonsymmetric matrix A, the
  eigenvalues and, optionally, the left and/or right eigenvectors. The right
  eigenvector v(j) of A satisfies

  A*v(j)= lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

  u(j)H*A = lambda(j)*u(j)H

  where u(j)H denotes the conjugate transpose of u(j). The computed
  eigenvectors are normalized to have Euclidean norm equal to 1 and
  largest component real.

  Example Program Results.
  ========================

 LAPACKE_zgeev (column-major, high-level) Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        MKL_Complex16 w[N], vl[LDVL*N], vr[LDVR*N];
        MKL_Complex16 a[LDA*N] = {
           {-3.84,  2.25}, {-0.66,  0.83}, {-3.99, -4.73}, { 7.74,  4.18},
           {-8.94, -4.75}, {-4.40, -3.82}, {-5.88, -6.60}, { 3.66, -7.53},
           { 8.95, -6.53}, {-3.50, -4.26}, {-3.36, -0.40}, { 2.58,  3.60},
```

```
           {-9.87,  4.82}, {-3.15,  7.36}, {-0.75,  5.23}, { 4.59,  5.41}
        };
        /* Executable statements */
        printf( "LAPACKE_zgeev (column-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zgeev( LAPACK_COL_MAJOR, 'V', 'V', n, a, lda, w, vl,
         ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_zgeev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}
```

LAPACKE_zgeev Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgeev Example.
   ======================

   Program computes the eigenvalues and left and right eigenvectors of a general
   rectangular matrix A:
```

```
   ( -3.84,  2.25) ( -8.94, -4.75) (  8.95, -6.53) ( -9.87,  4.82)
   ( -0.66,  0.83) ( -4.40, -3.82) ( -3.50, -4.26) ( -3.15,  7.36)
   ( -3.99, -4.73) ( -5.88, -6.60) ( -3.36, -0.40) ( -0.75,  5.23)
   (  7.74,  4.18) (  3.66, -7.53) (  2.58,  3.60) (  4.59,  5.41)

  Description.
  ============

  The routine computes for an n-by-n complex nonsymmetric matrix A, the
  eigenvalues and, optionally, the left and/or right eigenvectors. The right
  eigenvector v(j) of A satisfies

  A*v(j)= lambda(j)*v(j)

  where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

  u(j)H*A = lambda(j)*u(j)H

  where u(j)H denotes the conjugate transpose of u(j). The computed
  eigenvectors are normalized to have Euclidean norm equal to 1 and
  largest component real.

  Example Program Results.
  ========================

 LAPACKE_zgeev (row-major, high-level) Example Program Results

 Eigenvalues
 ( -9.43,-12.98) ( -3.44, 12.69) (  0.11, -3.40) (  5.76,  7.13)

 Left eigenvectors
 (  0.24, -0.18) (  0.61,  0.00) ( -0.18, -0.33) (  0.28,  0.09)
 (  0.79,  0.00) ( -0.05, -0.27) (  0.82,  0.00) ( -0.55,  0.16)
 (  0.22, -0.27) ( -0.21,  0.53) ( -0.37,  0.15) (  0.45,  0.09)
 ( -0.02,  0.41) (  0.40, -0.24) (  0.06,  0.12) (  0.62,  0.00)

 Right eigenvectors
 (  0.43,  0.33) (  0.83,  0.00) (  0.60,  0.00) ( -0.31,  0.03)
 (  0.51, -0.03) (  0.08, -0.25) ( -0.40, -0.20) (  0.04,  0.34)
 (  0.62,  0.00) ( -0.25,  0.28) ( -0.09, -0.48) (  0.36,  0.06)
 ( -0.23,  0.11) ( -0.10, -0.32) ( -0.43,  0.13) (  0.81,  0.00)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );

/* Parameters */
#define N 4
#define LDA N
#define LDVL N
#define LDVR N

/* Main program */
int main() {
        /* Locals */
```

```
        MKL_INT n = N, lda = LDA, ldvl = LDVL, ldvr = LDVR, info;
        /* Local arrays */
        MKL_Complex16 w[N], vl[LDVL*N], vr[LDVR*N];
        MKL_Complex16 a[LDA*N] = {
           {-3.84,  2.25}, {-8.94, -4.75}, { 8.95, -6.53}, {-9.87,  4.82},
           {-0.66,  0.83}, {-4.40, -3.82}, {-3.50, -4.26}, {-3.15,  7.36},
           {-3.99, -4.73}, {-5.88, -6.60}, {-3.36, -0.40}, {-0.75,  5.23},
           { 7.74,  4.18}, { 3.66, -7.53}, { 2.58,  3.60}, { 4.59,  5.41}
        };
        /* Executable statements */
        printf( "LAPACKE_zgeev (row-major, high-level) Example Program Results\n" );
        /* Solve eigenproblem */
        info = LAPACKE_zgeev( LAPACK_ROW_MAJOR, 'V', 'V', n, a, lda, w, vl,
         ldvl, vr, ldvr );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm failed to compute eigenvalues.\n" );
                exit( 1 );
        }
        /* Print eigenvalues */
        print_matrix( "Eigenvalues", 1, n, w, 1 );
        /* Print left eigenvectors */
        print_matrix( "Left eigenvectors", n, n, vl, ldvl );
        /* Print right eigenvectors */
        print_matrix( "Right eigenvectors", n, n, vr, ldvr );
        exit( 0 );
} /* End of LAPACKE_zgeev Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}
```

Singular Value Decomposition

Routine	Description	Examples
?gesvd	Computes the singular value decomposition of a general rectangular matrix.	cgesvd dgesvd sgesvd zgesvd
?gesdd	Computes the singular value decomposition of a general rectangular matrix using a divide and conquer algorithm.	cgesdd dgesdd sgesdd zgesdd

?gesvd function

Computes the singular value decomposition of a general rectangular matrix.

NOTE
For a detailed description and reference information on this function, please visit:

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call dgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call cgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info)
call zgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info)
```

Calling from C:

```
sgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info);
dgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info);
cgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info);
zgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info);
```

C Interface

```
lapack_int LAPACKE_sgesvd(int matrix_layout, char jobu, char jobvt, lapack_int m,
lapack_int n, float* a, lapack_int lda, float* s, float* u, lapack_int ldu, float* vt,
lapack_int ldvt, float* superb);
lapack_int LAPACKE_dgesvd(int matrix_layout, char jobu, char jobvt, lapack_int m,
lapack_int n, double* a, lapack_int lda, double* s, double* u, lapack_int ldu, double* vt,
lapack_int ldvt, double* superb);
lapack_int LAPACKE_cgesvd(int matrix_layout, char jobu, char jobvt, lapack_int m,
lapack_int n, lapack_complex_float* a, lapack_int lda, float* s, lapack_complex_float* u,
lapack_int ldu, lapack_complex_float* vt, lapack_int ldvt, float* superb);
lapack_int LAPACKE_zgesvd(int matrix_layout, char jobu, char jobvt, lapack_int m,
lapack_int n, lapack_complex_double* a, lapack_int lda, double* s, lapack_complex_double*
u, lapack_int ldu, lapack_complex_double* vt, lapack_int ldvt, double* superb);
```

See Also

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran
Notational Conventions
C Interface Conventions

CGESVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular complex matrix *A*, optionally the left and/or right singular vectors.

The SVD is written as:

```
A = U*SIGMA*V^H
```

where

- *A* is a complex *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its `min(m,n)` diagonal elements.
- *U* is an *m*-by-*m* unitary matrix.
- V^H (*V* conjugate transposed) is an *n*-by-*n* unitary matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first `min(m,n)` columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^H, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGESVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CGESVD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular complex matrix A:
*
*  (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
*  ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
*  ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)
*
*  Description.
```

```
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a complex
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. The SVD is written as
*
*  A = U*SIGMA*VH
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
*  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VH, not V.
*
*  Example Program Results.
*  ========================
*
* CGESVD Example Program Results
*
* Singular values
*  17.63  11.61   6.78
*
* Left singular vectors (stored columnwise)
* ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
* ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
* (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)
*
* Right singular vectors (stored rowwise)
* ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
* (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
* (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 3, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least MAX( 1, 5*MIN(M,N) )
      REAL             S( M ), RWORK( 5*M )
      COMPLEX          A( LDA, N ), U( LDU, M ), VT( LDVT, N ),
     $                 WORK( LWMAX )
      DATA             A/
     $ ( 5.91,-5.69),(-3.15,-4.08),(-4.89, 4.20),
     $ ( 7.09, 2.72),(-1.89, 3.27),( 4.10,-6.70),
     $ ( 7.78,-4.06),( 4.57,-2.07),( 3.28,-3.84),
     $ (-0.79,-7.21),(-3.88,-3.30),( 3.84, 1.19)
     $                  /
```

```
*
*     .. External Subroutines ..
      EXTERNAL         CGESVD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CGESVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, RWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL CGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, RWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, M, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   M, N, VT, LDVT )
      STOP
      END
*
*     End of CGESVD Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
```

```
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CGESVD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   CGESVD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
```

```
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 CGESVD Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGESVD prototype */
extern void cgesvd( char* jobu, char* jobvt, int* m, int* n, fcomplex* a,
                int* lda, float* s, fcomplex* u, int* ldu, fcomplex* vt, int* ldvt,
                fcomplex* work, int* lwork, float* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
```

```
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least max( 1, 5*min(m,n) ) */
        float s[M], rwork[5*M];
        fcomplex u[LDU*M], vt[LDVT*N];
        fcomplex a[LDA*N] = {
           { 5.91f, -5.69f}, {-3.15f, -4.08f}, {-4.89f,  4.20f},
           { 7.09f,  2.72f}, {-1.89f,  3.27f}, { 4.10f, -6.70f},
           { 7.78f, -4.06f}, { 4.57f, -2.07f}, { 3.28f, -3.84f},
           {-0.79f, -7.21f}, {-3.88f, -3.30f}, { 3.84f,  1.19f}
        };
        /* Executable statements */
        printf( " CGESVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt, &lwork,
         rwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Compute SVD */
        cgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work, &lwork,
         rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CGESVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
```

```
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cgesvd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesvd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================
```

```
 LAPACKE_cgesvd (row-major, high-level) Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        float s[M];
        float superb[min(M,N)-1];
        MKL_Complex8 u[LDU*M], vt[LDVT*N];
        MKL_Complex8 a[LDA*M] = {
           { 5.91f, -5.69f}, { 7.09f,  2.72f}, { 7.78f, -4.06f}, {-0.79f, -7.21f},
           {-3.15f, -4.08f}, {-1.89f,  3.27f}, { 4.57f, -2.07f}, {-3.88f, -3.30f},
           {-4.89f,  4.20f}, { 4.10f, -6.70f}, { 3.28f, -3.84f}, { 3.84f,  1.19f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesvd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_cgesvd( LAPACK_ROW_MAJOR, 'A', 'A', m, n, a, lda, s,
         u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
```

```
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_cgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cgesvd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesvd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)
```

```
   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_cgesvd (column-major, high-level) Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
```

```
        float s[M];
        float superb[min(M,N)-1];
        MKL_Complex8 u[LDU*M], vt[LDVT*N];
        MKL_Complex8 a[LDA*N] = {
           { 5.91f, -5.69f}, {-3.15f, -4.08f}, {-4.89f,  4.20f},
           { 7.09f,  2.72f}, {-1.89f,  3.27f}, { 4.10f, -6.70f},
           { 7.78f, -4.06f}, { 4.57f, -2.07f}, { 3.28f, -3.84f},
           {-0.79f, -7.21f}, {-3.88f, -3.30f}, { 3.84f,  1.19f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesvd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_cgesvd( LAPACK_COL_MAJOR, 'A', 'A', m, n, a, lda, s,
         u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_cgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

DGESVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular real matrix *A*, optionally the left and/or right singular vectors.

The SVD is written as:

```
A = U*SIGMA*V^T
```

where

- *A* is a real *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its `min(m,n)` diagonal elements.
- *U* is an *m*-by-*m* orthogonal matrix.
- V^T (*V* transposed) is an *n*-by-*n* orthogonal matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first `min(m,n)` columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^T, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGESVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  DGESVD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular matrix A:
*
*    8.79   9.93   9.83   5.45   3.16
*    6.11   6.91   5.04  -0.27   7.98
*   -9.15  -7.93   4.86   4.85   3.01
```

```
*    9.57   1.64   8.83   0.74   5.80
*   -3.49   4.02   9.80  10.00   4.27
*    9.84   0.15  -8.99  -6.02  -5.31
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a real
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. The SVD is written as
*
*  A = U*SIGMA*VT
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
*  is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VT, not V.
*
*  Example Program Results.
*  ========================
*
* DGESVD Example Program Results
*
* Singular values
*  27.47  22.64   8.56   5.99   2.01
*
* Left singular vectors (stored columnwise)
*  -0.59   0.26   0.36   0.31   0.23
*  -0.40   0.24  -0.22  -0.75  -0.36
*  -0.03  -0.60  -0.45   0.23  -0.31
*  -0.43   0.24  -0.69   0.33   0.16
*  -0.47  -0.35   0.39   0.16  -0.52
*   0.29   0.58  -0.02   0.38  -0.65
*
* Right singular vectors (stored rowwise)
*  -0.25  -0.40  -0.69  -0.37  -0.41
*   0.81   0.36  -0.25  -0.37  -0.10
*  -0.26   0.70  -0.22   0.39  -0.49
*   0.40  -0.45   0.25   0.43  -0.62
*  -0.22   0.14   0.59  -0.63  -0.44
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 6, N = 5 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
```

```
      DOUBLE PRECISION A( LDA, N ), U( LDU, M ), VT( LDVT, N ), S( N ),
     $                 WORK( LWMAX )
      DATA             A/
     $  8.79, 6.11,-9.15, 9.57,-3.49, 9.84,
     $  9.93, 6.91,-7.93, 1.64, 4.02, 0.15,
     $  9.83, 5.04, 4.86, 8.83, 9.80,-8.99,
     $  5.45,-0.27, 4.85, 0.74,10.00,-6.02,
     $  3.16, 7.98, 3.01, 5.80, 4.27,-5.31
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DGESVD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DGESVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL DGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, N, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, N, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   N, N, VT, LDVT )
      STOP
      END
*
*     End of DGESVD Example.
*
*  ============================================================================
```

```
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DGESVD Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   DGESVD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
```

```
   vectors. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 DGESVD Example Program Results

 Singular values
  27.47  22.64   8.56   5.99   2.01

 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>

/* DGESVD prototype */
extern void dgesvd( char* jobu, char* jobvt, int* m, int* n, double* a,
                int* lda, double* s, double* u, int* ldu, double* vt, int* ldvt,
                double* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 6
#define N 5
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        double wkopt;
```

```
        double* work;
        /* Local arrays */
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*N] = {
            8.79,  6.11, -9.15,  9.57, -3.49,  9.84,
            9.93,  6.91, -7.93,  1.64,  4.02,  0.15,
            9.83,  5.04,  4.86,  8.83,  9.80, -8.99,
            5.45, -0.27,  4.85,  0.74, 10.00, -6.02,
            3.16,  7.98,  3.01,  5.80,  4.27, -5.31
        };
        /* Executable statements */
        printf( " DGESVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt, &lwork,
         &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Compute SVD */
        dgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work, &lwork,
         &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DGESVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dgesvd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
```

```
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesvd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_dgesvd (row-major, high-level) Example Program Results

 Singular values
  27.47  22.64   8.56   5.99   2.01

 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
```

```
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 5
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        double superb[min(M,N)-1];
        /* Local arrays */
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*M] = {
            8.79,  9.93,  9.83, 5.45,  3.16,
            6.11,  6.91,  5.04, -0.27,  7.98,
           -9.15, -7.93,  4.86, 4.85,  3.01,
            9.57,  1.64,  8.83, 0.74,  5.80,
           -3.49,  4.02,  9.80, 10.00,  4.27,
            9.84,  0.15, -8.99, -6.02, -5.31
        };
        /* Executable statements */
        printf( "LAPACKE_dgesvd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_dgesvd( LAPACK_ROW_MAJOR, 'A', 'A', m, n, a, lda,
                        s, u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_dgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
```

```
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dgesvd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesvd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.
```

```
   Example Program Results.
   ========================

 LAPACKE_dgesvd (column-major, high-level) Example Program Results

 Singular values
  27.47  22.64   8.56   5.99   2.01

 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 5
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        double superb[min(M,N)-1];
        /* Local arrays */
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*N] = {
            8.79,  6.11, -9.15,  9.57, -3.49,  9.84,
            9.93,  6.91, -7.93,  1.64,  4.02,  0.15,
            9.83,  5.04,  4.86,  8.83,  9.80, -8.99,
            5.45, -0.27,  4.85,  0.74, 10.00, -6.02,
            3.16,  7.98,  3.01,  5.80,  4.27, -5.31
        };
        /* Executable statements */
        printf( "LAPACKE_dgesvd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_dgesvd( LAPACK_COL_MAJOR, 'A', 'A', m, n, a, lda,
                        s, u, ldu, vt, ldvt, superb );
```

```
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_dgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SGESVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular real matrix *A*, optionally the left and/or right singular vectors.

The SVD is written as:

$$A = U*SIGMA*V^T$$

where

- *A* is a real *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its $\min(m,n)$ diagonal elements.
- *U* is an *m*-by-*m* orthogonal matrix.
- V^T (*V* transposed) is an *n*-by-*n* orthogonal matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^T, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGESVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SGESVD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular matrix A:
*
*    8.79   9.93   9.83   5.45   3.16
*    6.11   6.91   5.04  -0.27   7.98
*   -9.15  -7.93   4.86   4.85   3.01
*    9.57   1.64   8.83   0.74   5.80
*   -3.49   4.02   9.80  10.00   4.27
*    9.84   0.15  -8.99  -6.02  -5.31
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a real
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. The SVD is written as
*
*  A = U*SIGMA*VT
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
*  is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VT, not V.
*
```

```
*  Example Program Results.
*  ========================
*
* SGESVD Example Program Results
*
* Singular values
*  27.47  22.64   8.56   5.99   2.01
*
* Left singular vectors (stored columnwise)
*  -0.59   0.26   0.36   0.31   0.23
*  -0.40   0.24  -0.22  -0.75  -0.36
*  -0.03  -0.60  -0.45   0.23  -0.31
*  -0.43   0.24  -0.69   0.33   0.16
*  -0.47  -0.35   0.39   0.16  -0.52
*   0.29   0.58  -0.02   0.38  -0.65
*
* Right singular vectors (stored rowwise)
*  -0.25  -0.40  -0.69  -0.37  -0.41
*   0.81   0.36  -0.25  -0.37  -0.10
*  -0.26   0.70  -0.22   0.39  -0.49
*   0.40  -0.45   0.25   0.43  -0.62
*  -0.22   0.14   0.59  -0.63  -0.44
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 6, N = 5 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
      REAL             A( LDA, N ), U( LDU, M ), VT( LDVT, N ), S( N ),
     $                 WORK( LWMAX )
      DATA             A/
     $  8.79, 6.11,-9.15, 9.57,-3.49, 9.84,
     $  9.93, 6.91,-7.93, 1.64, 4.02, 0.15,
     $  9.83, 5.04, 4.86, 8.83, 9.80,-8.99,
     $  5.45,-0.27, 4.85, 0.74,10.00,-6.02,
     $  3.16, 7.98, 3.01, 5.80, 4.27,-5.31
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SGESVD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SGESVD Example Program Results'
*
*     Query the optimal workspace.
*
```

```
      LWORK = -1
      CALL SGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL SGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, N, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, N, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   N, N, VT, LDVT )
      STOP
      END
*
*     End of SGESVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SGESVD Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   SGESVD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

SGESVD Example Program Results

Singular values
 27.47  22.64   8.56   5.99   2.01
```

```
 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>

/* SGESVD prototype */
extern void sgesvd( char* jobu, char* jobvt, int* m, int* n, float* a,
                int* lda, float* s, float* u, int* ldu, float* vt, int* ldvt,
                float* work, int* lwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 6
#define N 5
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*N] = {
            8.79f,  6.11f, -9.15f,  9.57f, -3.49f,  9.84f,
            9.93f,  6.91f, -7.93f,  1.64f,  4.02f,  0.15f,
            9.83f,  5.04f,  4.86f,  8.83f,  9.80f, -8.99f,
            5.45f, -0.27f,  4.85f,  0.74f, 10.00f, -6.02f,
            3.16f,  7.98f,  3.01f,  5.80f,  4.27f, -5.31f
        };
        /* Executable statements */
        printf( " SGESVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        sgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt, &lwork,
         &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Compute SVD */
        sgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work, &lwork,
         &info );
```

```
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SGESVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_sgesvd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgesvd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31
```

```
  Description.
  ============

  The routine computes the singular value decomposition (SVD) of a real
  m-by-n matrix A, optionally computing the left and/or right singular
  vectors. The SVD is written as

  A = U*SIGMA*VT

  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
  diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
  is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
  are the singular values of A; they are real and non-negative, and are
  returned in descending order. The first min(m, n) columns of U and V are
  the left and right singular vectors of A.

  Note that the routine returns VT, not V.

  Example Program Results.
  ========================

 LAPACKE_sgesvd (row-major, high-level) Example Program Results

 Singular values
  27.47  22.64   8.56   5.99   2.01

 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 5
#define LDA N
#define LDU M
#define LDVT N
```

```
/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        float superb[min(M,N)-1];
        /* Local arrays */
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*M] = {
            8.79f,  9.93f,  9.83f, 5.45f,  3.16f,
            6.11f,  6.91f,  5.04f, -0.27f,  7.98f,
           -9.15f, -7.93f,  4.86f, 4.85f,  3.01f,
            9.57f,  1.64f,  8.83f, 0.74f,  5.80f,
           -3.49f,  4.02f,  9.80f, 10.00f,  4.27f,
            9.84f,  0.15f, -8.99f, -6.02f, -5.31f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesvd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_sgesvd( LAPACK_ROW_MAJOR, 'A', 'A', m, n, a, lda,
                        s, u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_sgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_sgesvd Example Program in C for Column Major Data Layout

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
```

```
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgesvd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular matrix A:

     8.79   9.93   9.83   5.45   3.16
     6.11   6.91   5.04  -0.27   7.98
    -9.15  -7.93   4.86   4.85   3.01
     9.57   1.64   8.83   0.74   5.80
    -3.49   4.02   9.80  10.00   4.27
     9.84   0.15  -8.99  -6.02  -5.31

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_sgesvd (column-major, high-level) Example Program Results

 Singular values
  27.47  22.64   8.56   5.99   2.01

 Left singular vectors (stored columnwise)
  -0.59   0.26   0.36   0.31   0.23
  -0.40   0.24  -0.22  -0.75  -0.36
  -0.03  -0.60  -0.45   0.23  -0.31
  -0.43   0.24  -0.69   0.33   0.16
  -0.47  -0.35   0.39   0.16  -0.52
   0.29   0.58  -0.02   0.38  -0.65

 Right singular vectors (stored rowwise)
  -0.25  -0.40  -0.69  -0.37  -0.41
   0.81   0.36  -0.25  -0.37  -0.10
  -0.26   0.70  -0.22   0.39  -0.49
   0.40  -0.45   0.25   0.43  -0.62
```

```
  -0.22   0.14   0.59  -0.63  -0.44
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 5
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        float superb[min(M,N)-1];
        /* Local arrays */
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*N] = {
            8.79f,  6.11f, -9.15f,  9.57f, -3.49f,  9.84f,
            9.93f,  6.91f, -7.93f,  1.64f,  4.02f,  0.15f,
            9.83f,  5.04f,  4.86f,  8.83f,  9.80f, -8.99f,
            5.45f, -0.27f,  4.85f,  0.74f, 10.00f, -6.02f,
            3.16f,  7.98f,  3.01f,  5.80f,  4.27f, -5.31f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesvd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_sgesvd( LAPACK_COL_MAJOR, 'A', 'A', m, n, a, lda,
                        s, u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_sgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
```

```
                printf( "\n" );
        }
}
```

ZGESVD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular complex matrix *A*, optionally the left and/or right singular vectors.

The SVD is written as:

$$A = U*SIGMA*V^H$$

where

- *A* is a complex *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its $\min(m,n)$ diagonal elements.
- *U* is an *m*-by-*m* unitary matrix.
- V^H (*V* conjugate transposed) is an *n*-by-*n* unitary matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^H, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesvd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGESVD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
```

```
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =============================================================================
*
*  ZGESVD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular complex matrix A:
*
*  (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
*  ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
*  ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a complex
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. The SVD is written as
*
*  A = U*SIGMA*VH
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
*  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VH, not V.
*
*  Example Program Results.
*  ========================
*
* ZGESVD Example Program Results
*
* Singular values
*  17.63  11.61   6.78
*
* Left singular vectors (stored columnwise)
* ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
* ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
* (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)
*
* Right singular vectors (stored rowwise)
* ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
* (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
* (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 3, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
```

```
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     RWORK dimension should be at least MAX( 1, 5*MIN(M,N) )
      DOUBLE PRECISION S( M ), RWORK( 5*M )
      COMPLEX*16       A( LDA, N ), U( LDU, M ), VT( LDVT, N ),
     $                 WORK( LWMAX )
      DATA             A/
     $ ( 5.91,-5.69),(-3.15,-4.08),(-4.89, 4.20),
     $ ( 7.09, 2.72),(-1.89, 3.27),( 4.10,-6.70),
     $ ( 7.78,-4.06),( 4.57,-2.07),( 3.28,-3.84),
     $ (-0.79,-7.21),(-3.88,-3.30),( 3.84, 1.19)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZGESVD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZGESVD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, RWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL ZGESVD( 'All', 'All', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $             WORK, LWORK, RWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, M, U, LDU )
*
*     Print right singular vectors.
*
```

```
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   M, N, VT, LDVT )
      STOP
      END
*
*     End of ZGESVD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZGESVD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
```

```
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZGESVD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 ZGESVD Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>
```

```
/* Complex datatype */
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGESVD prototype */
extern void zgesvd( char* jobu, char* jobvt, int* m, int* n, dcomplex* a,
                int* lda, double* s, dcomplex* u, int* ldu, dcomplex* vt, int* ldvt,
                dcomplex* work, int* lwork, double* rwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* rwork dimension should be at least max( 1, 5*min(m,n) ) */
        double s[M], rwork[5*M];
        dcomplex u[LDU*M], vt[LDVT*N];
        dcomplex a[LDA*N] = {
           { 5.91, -5.69}, {-3.15, -4.08}, {-4.89,  4.20},
           { 7.09,  2.72}, {-1.89,  3.27}, { 4.10, -6.70},
           { 7.78, -4.06}, { 4.57, -2.07}, { 3.28, -3.84},
           {-0.79, -7.21}, {-3.88, -3.30}, { 3.84,  1.19}
        };
        /* Executable statements */
        printf( " ZGESVD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt, &lwork,
         rwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Compute SVD */
        zgesvd( "All", "All", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work, &lwork,
         rwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        /* Free workspace */
```

```
        free( (void*)work );
        exit( 0 );
} /* End of ZGESVD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zgesvd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesvd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)

   Description.
   ============
```

```
   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_zgesvd (row-major, high-level) Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[M];
        double superb[min(M,N)-1];
        MKL_Complex16 u[LDU*M], vt[LDVT*N];
```

```
        MKL_Complex16 a[LDA*M] = {
           { 5.91, -5.69}, { 7.09,  2.72}, { 7.78, -4.06}, {-0.79, -7.21},
           {-3.15, -4.08}, {-1.89,  3.27}, { 4.57, -2.07}, {-3.88, -3.30},
           {-4.89,  4.20}, { 4.10, -6.70}, { 3.28, -3.84}, { 3.84,  1.19}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesvd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_zgesvd( LAPACK_ROW_MAJOR, 'A', 'A', m, n, a, lda, s,
         u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_zgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_zgesvd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
```

```
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesvd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A:

   (  5.91, -5.69) (  7.09,  2.72) (  7.78, -4.06) ( -0.79, -7.21)
   ( -3.15, -4.08) ( -1.89,  3.27) (  4.57, -2.07) ( -3.88, -3.30)
   ( -4.89,  4.20) (  4.10, -6.70) (  3.28, -3.84) (  3.84,  1.19)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_zgesvd (column-major, high-level) Example Program Results

 Singular values
  17.63  11.61   6.78

 Left singular vectors (stored columnwise)
 ( -0.86,  0.00) (  0.40,  0.00) (  0.32,  0.00)
 ( -0.35,  0.13) ( -0.24, -0.21) ( -0.63,  0.60)
 (  0.15,  0.32) (  0.61,  0.61) ( -0.36,  0.10)

 Right singular vectors (stored rowwise)
 ( -0.22,  0.51) ( -0.37, -0.32) ( -0.53,  0.11) (  0.15,  0.38)
 (  0.31,  0.31) (  0.09, -0.57) (  0.18, -0.39) (  0.38, -0.39)
 (  0.53,  0.24) (  0.49,  0.28) ( -0.47, -0.25) ( -0.15,  0.19)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

#define min(a,b) ((a)>(b)?(b):(a))
```

```
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[M];
        double superb[min(M,N)-1];
        MKL_Complex16 u[LDU*M], vt[LDVT*N];
        MKL_Complex16 a[LDA*N] = {
           { 5.91, -5.69}, {-3.15, -4.08}, {-4.89,  4.20},
           { 7.09,  2.72}, {-1.89,  3.27}, { 4.10, -6.70},
           { 7.78, -4.06}, { 4.57, -2.07}, { 3.28, -3.84},
           {-0.79, -7.21}, {-3.88, -3.30}, { 3.84,  1.19}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesvd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_zgesvd( LAPACK_COL_MAJOR, 'A', 'A', m, n, a, lda, s,
         u, ldu, vt, ldvt, superb );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_zgesvd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
```

```
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

?gesdd function

Computes the singular value decomposition of a general rectangular matrix using a divide and conquer method.

NOTE
For a detailed description and reference information on this function, please visit:

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

Fortran-77 Interface

Calling from Fortran:

```
call sgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info)
call dgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info)
call cgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork, info)
call zgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork, info)
```

Calling from C:

```
sgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info);
dgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info);
cgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork, info);
zgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork, info);
```

C Interface

```
lapack_int LAPACKE_sgesdd(int matrix_layout, char jobz, lapack_int m, lapack_int n, float*
a, lapack_int lda, float* s, float* u, lapack_int ldu, float* vt, lapack_int ldvt);
lapack_int LAPACKE_dgesdd(int matrix_layout, char jobz, lapack_int m, lapack_int n, double*
a, lapack_int lda, double* s, double* u, lapack_int ldu, double* vt, lapack_int ldvt);
lapack_int LAPACKE_cgesdd(int matrix_layout, char jobz, lapack_int m, lapack_int n,
lapack_complex_float* a, lapack_int lda, float* s, lapack_complex_float* u, lapack_int ldu,
lapack_complex_float* vt, lapack_int ldvt);
lapack_int LAPACKE_zgesdd(int matrix_layout, char jobz, lapack_int m, lapack_int n,
lapack_complex_double* a, lapack_int lda, double* s, lapack_complex_double* u, lapack_int
ldu, lapack_complex_double* vt, lapack_int ldvt);
```

See Also

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGESDD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular complex matrix *A*, optionally the left and/or right singular vectors. This routine uses a divide and conquer algorithm to compute SVD.

The SVD is written as

```
A = U*SIGMA*V^H
```

where

- *A* is a complex *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its `min(m,n)` diagonal elements.
- *U* is an *m*-by-*m* unitary matrix.
- V^H (*V* conjugate transposed) is an *n*-by-*n* unitary matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first `min(m,n)` columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^H, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

CGESDD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  CGESDD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular complex matrix A using a divide and conquer method, where A is:
*
*  ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
*  (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
*  (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)
```

```
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a complex
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. If singular vectors are desired, it uses a divide and conquer
*  algorithm. The SVD is written as
*
*  A = U*SIGMA*VH
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
*  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VH, not V.
*
*  Example Program Results.
*  ========================
*
* CGESDD Example Program Results
*
* Singular values
*  21.76  16.60   3.97
*
* Left singular vectors (stored columnwise)
* (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
* ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
* (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)
*
* Right singular vectors (stored rowwise)
* (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
* ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
* (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*  =============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 3, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 8*MIN(M,N)
      INTEGER          IWORK( 8*M )
*     RWORK dimension should be at least 5*(MIN(M,N))**2 + 7*MIN(M,N))
      REAL             S( M ), RWORK( 5*M*M + 7*M )
      COMPLEX          A( LDA, N ), U( LDU, M ), VT( LDVT, N ),
     $                 WORK( LWMAX )
      DATA             A/
```

```
     $ (-5.40, 7.40),( 1.09, 1.55),( 9.88, 1.91),
     $ ( 6.00, 6.38),( 2.60, 0.07),( 4.92, 6.31),
     $ ( 9.91, 0.16),( 3.98,-5.26),(-2.11, 7.39),
     $ (-5.28,-4.16),( 2.03, 1.11),(-9.81,-8.98)
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         CGESDD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'CGESDD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL CGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, RWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL CGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, RWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, M, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   M, N, VT, LDVT )
      STOP
      END
*
*     End of CGESDD Example.
*
*  ============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
```

```
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX          A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

CGESDD Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   CGESDD Example.
   ==============
```

```
  Program computes the singular value decomposition of a general
  rectangular complex matrix A using a divide and conquer method, where A is:

  ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
  (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
  (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

  Description.
  ============

  The routine computes the singular value decomposition (SVD) of a complex
  m-by-n matrix A, optionally computing the left and/or right singular
  vectors. If singular vectors are desired, it uses a divide and conquer
  algorithm. The SVD is written as

  A = U*SIGMA*VH

  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
  are the singular values of A; they are real and non-negative, and are
  returned in descending order. The first min(m, n) columns of U and V are
  the left and right singular vectors of A.

  Note that the routine returns VH, not V.

  Example Program Results.
  ========================

 CGESDD Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
struct _fcomplex { float re, im; };
typedef struct _fcomplex fcomplex;

/* CGESDD prototype */
extern void cgesdd( char* jobz, int* m, int* n, fcomplex* a,
                int* lda, float* s, fcomplex* u, int* ldu, fcomplex* vt, int* ldvt,
                fcomplex* work, int* lwork, float* rwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, fcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, float* a, int lda );
```

```
/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        fcomplex wkopt;
        fcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 8*min(m,n) */
        int iwork[8*M];
        /* rwork dimension should be at least 5*(min(m,n))**2 + 7*min(m,n)) */
        float s[M], rwork[5*M*M+7*M];
        fcomplex u[LDU*M], vt[LDVT*N];
        fcomplex a[LDA*N] = {
           {-5.40f,  7.40f}, { 1.09f,  1.55f}, { 9.88f,  1.91f},
           { 6.00f,  6.38f}, { 2.60f,  0.07f}, { 4.92f,  6.31f},
           { 9.91f,  0.16f}, { 3.98f, -5.26f}, {-2.11f,  7.39f},
           {-5.28f, -4.16f}, { 2.03f,  1.11f}, {-9.81f, -8.98f}
        };
        /* Executable statements */
        printf( " CGESDD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        cgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt,
         &lwork, rwork, iwork, &info );
        lwork = (int)wkopt.re;
        work = (fcomplex*)malloc( lwork*sizeof(fcomplex) );
        /* Compute SVD */
        cgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work,
         &lwork, rwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of CGESDD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, fcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_cgesdd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesdd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A using a divide and conquer method, where A is:

   ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
   (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
   (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
```

```
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_cgesdd (row-major, high-level) Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        float s[M];
        MKL_Complex8 u[LDU*M], vt[LDVT*N];
        MKL_Complex8 a[LDA*M] = {
           {-5.40f,  7.40f}, { 6.00f,  6.38f}, { 9.91f,  0.16f}, {-5.28f, -4.16f},
           { 1.09f,  1.55f}, { 2.60f,  0.07f}, { 3.98f, -5.26f}, { 2.03f,  1.11f},
           { 9.88f,  1.91f}, { 4.92f,  6.31f}, {-2.11f,  7.39f}, {-9.81f, -8.98f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesdd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_cgesdd( LAPACK_ROW_MAJOR, 'S', m, n, a, lda, s, u,
         ldu, vt, ldvt );
        /* Check for convergence */
```

```
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_cgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_cgesdd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_cgesdd Example.
   ======================

   Program computes the singular value decomposition of a general
```

```
   rectangular complex matrix A using a divide and conquer method, where A is:

   ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
   (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
   (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_cgesdd (column-major, high-level) Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N
```

```
/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        float s[M];
        MKL_Complex8 u[LDU*M], vt[LDVT*N];
        MKL_Complex8 a[LDA*N] = {
           {-5.40f,  7.40f}, { 1.09f,  1.55f}, { 9.88f,  1.91f},
           { 6.00f,  6.38f}, { 2.60f,  0.07f}, { 4.92f,  6.31f},
           { 9.91f,  0.16f}, { 3.98f, -5.26f}, {-2.11f,  7.39f},
           {-5.28f, -4.16f}, { 2.03f,  1.11f}, {-9.81f, -8.98f}
        };
        /* Executable statements */
        printf( "LAPACKE_cgesdd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_cgesdd( LAPACK_COL_MAJOR, 'S', m, n, a, lda, s, u,
         ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_cgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex8* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

DGESDD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular real matrix *A*, optionally the left and/or right singular vectors. This routine uses a divide and conquer algorithm to compute SVD.

The SVD is written as:

$$A = U*SIGMA*V^T$$

where

- *A* is a real *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its $\min(m,n)$ diagonal elements.
- *U* is an *m*-by-*m* orthogonal matrix.
- V^T (*V* transposed) is an *n*-by-*n* orthogonal matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^T, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

DGESDD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
```

```
*
*  DGESDD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular matrix A using a divide and conquer method, where A is:
*
*    7.52  -1.10  -7.95   1.08
*   -0.76   0.62   9.34  -7.10
*    5.13   6.62  -5.66   0.87
*   -4.75   8.52   5.75   5.30
*    1.33   4.91  -5.49  -3.52
*   -2.40  -6.77   2.34   3.95
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a real
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. If singular vectors are desired, it uses a divide and conquer
*  algorithm. The SVD is written as
*
*  A = U*SIGMA*VT
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
*  is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VT, not V.
*
*  Example Program Results.
*  ========================
*
* DGESDD Example Program Results
*
* Singular values
*  18.37  13.63  10.85   4.49
*
* Left singular vectors (stored columnwise)
*  -0.57   0.18   0.01   0.53
*   0.46  -0.11  -0.72   0.42
*  -0.45  -0.41   0.00   0.36
*   0.33  -0.69   0.49   0.19
*  -0.32  -0.31  -0.28  -0.61
*   0.21   0.46   0.39   0.09
*
* Right singular vectors (stored rowwise)
*  -0.52  -0.12   0.85  -0.03
*   0.08  -0.99  -0.09  -0.01
*  -0.28  -0.02  -0.14   0.95
*   0.81   0.01   0.50   0.31
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
```

```
      PARAMETER        ( M = 6, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 8*MIN(M,N)
      INTEGER          IWORK( 8*N )
      DOUBLE PRECISION A( LDA, N ), U( LDU, M ), VT( LDVT, N ), S( N ),
     $                 WORK( LWMAX )
      DATA             A/
     $  7.52,-0.76, 5.13,-4.75, 1.33,-2.40,
     $ -1.10, 0.62, 6.62, 8.52, 4.91,-6.77,
     $ -7.95, 9.34,-5.66, 5.75,-5.49, 2.34,
     $  1.08,-7.10, 0.87, 5.30,-3.52, 3.95
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         DGESDD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'DGESDD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL DGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL DGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, N, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, N, U, LDU )
```

```
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   N, N, VT, LDVT )
      STOP
      END
*
*     End of DGESDD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

DGESDD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   DGESDD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
```

```
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 DGESDD Example Program Results

 Singular values
  18.37  13.63  10.85   4.49

 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>

/* DGESDD prototype */
extern void dgesdd( char* jobz, int* m, int* n, double* a,
                int* lda, double* s, double* u, int* ldu, double* vt, int* ldvt,
                double* work, int* lwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
```

```
#define M 6
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        double wkopt;
        double* work;
        /* Local arrays */
   /* iwork dimension should be at least 8*min(m,n) */
   int iwork[8*N];
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*N] = {
            7.52, -0.76,  5.13, -4.75,  1.33, -2.40,
           -1.10,  0.62,  6.62,  8.52,  4.91, -6.77,
           -7.95,  9.34, -5.66,  5.75, -5.49,  2.34,
            1.08, -7.10,  0.87,  5.30, -3.52,  3.95
        };
        /* Executable statements */
        printf( " DGESDD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        dgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt,
         &lwork, iwork, &info );
        lwork = (int)wkopt;
        work = (double*)malloc( lwork*sizeof(double) );
        /* Compute SVD */
        dgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work,
         &lwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of DGESDD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_dgesdd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesdd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_dgesdd (row-major, high-level) Example Program Results

 Singular values
  18.37  13.63  10.85   4.49
```

```
 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*M] = {
            7.52, -1.10, -7.95, 1.08,
           -0.76,  0.62,  9.34, -7.10,
            5.13,  6.62, -5.66, 0.87,
           -4.75,  8.52,  5.75, 5.30,
            1.33,  4.91, -5.49, -3.52,
           -2.40, -6.77,  2.34, 3.95
        };
        /* Executable statements */
        printf( "LAPACKE_dgesdd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_dgesdd( LAPACK_ROW_MAJOR, 'S', m, n, a, lda, s,
                        u, ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
```

```
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_dgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_dgesdd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_dgesdd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
```

```
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_dgesdd (column-major, high-level) Example Program Results

 Singular values
  18.37  13.63  10.85   4.49

 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[N], u[LDU*M], vt[LDVT*N];
        double a[LDA*N] = {
            7.52, -0.76,  5.13, -4.75,  1.33, -2.40,
           -1.10,  0.62,  6.62,  8.52,  4.91, -6.77,
           -7.95,  9.34, -5.66,  5.75, -5.49,  2.34,
            1.08, -7.10,  0.87,  5.30, -3.52,  3.95
        };
        /* Executable statements */
        printf( "LAPACKE_dgesdd (column-major, high-level) Example Program Results\n" );
```

```
        /* Compute SVD */
        info = LAPACKE_dgesdd( LAPACK_COL_MAJOR, 'S', m, n, a, lda, s,
                        u, ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_dgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

SGESDD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular real matrix *A*, optionally the left and/or right singular vectors. This routine uses a divide and conquer algorithm to compute the SVD.

The SVD is written as:

$$A = U*SIGMA*V^T$$

where

- *A* is a real *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its $\min(m,n)$ diagonal elements.
- *U* is an *m*-by-*m* orthogonal matrix.
- V^T (*V* transposed) is an *n*-by-*n* orthogonal matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^T, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C
?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

SGESDD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
*
*  SGESDD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular matrix A using a divide and conquer method, where A is:
*
*    7.52  -1.10  -7.95   1.08
*   -0.76   0.62   9.34  -7.10
*    5.13   6.62  -5.66   0.87
*   -4.75   8.52   5.75   5.30
*    1.33   4.91  -5.49  -3.52
*   -2.40  -6.77   2.34   3.95
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a real
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. If singular vectors are desired, it uses a divide and conquer
*  algorithm. The SVD is written as
*
*  A = U*SIGMA*VT
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
*  is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VT, not V.
```

```
*
*  Example Program Results.
*  ========================
*
* SGESDD Example Program Results
*
* Singular values
*  18.37  13.63  10.85   4.49
*
* Left singular vectors (stored columnwise)
*  -0.57   0.18   0.01   0.53
*   0.46  -0.11  -0.72   0.42
*  -0.45  -0.41   0.00   0.36
*   0.33  -0.69   0.49   0.19
*  -0.32  -0.31  -0.28  -0.61
*   0.21   0.46   0.39   0.09
*
* Right singular vectors (stored rowwise)
*  -0.52  -0.12   0.85  -0.03
*   0.08  -0.99  -0.09  -0.01
*  -0.28  -0.02  -0.14   0.95
*   0.81   0.01   0.50   0.31
*  ============================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 6, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 8*MIN(M,N)
      INTEGER          IWORK( 8*N )
      REAL             A( LDA, N ), U( LDU, M ), VT( LDVT, N ), S( N ),
     $                 WORK( LWMAX )
      DATA             A/
     $  7.52,-0.76, 5.13,-4.75, 1.33,-2.40,
     $ -1.10, 0.62, 6.62, 8.52, 4.91,-6.77,
     $ -7.95, 9.34,-5.66, 5.75,-5.49, 2.34,
     $  1.08,-7.10, 0.87, 5.30,-3.52, 3.95
     $                  /
*
*     .. External Subroutines ..
      EXTERNAL         SGESDD
      EXTERNAL         PRINT_MATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'SGESDD Example Program Results'
*
*     Query the optimal workspace.
```

```
*
      LWORK = -1
      CALL SGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL SGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_MATRIX( 'Singular values', 1, N, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, N, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   N, N, VT, LDVT )
      STOP
      END
*
*     End of SGESDD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      REAL             A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

SGESDD Example Program in C

```
/*******************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*******************************************************************************
*/
/*
   SGESDD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 SGESDD Example Program Results

 Singular values
  18.37  13.63  10.85   4.49
```

```
 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>

/* SGESDD prototype */
extern void sgesdd( char* jobz, int* m, int* n, float* a,
                int* lda, float* s, float* u, int* ldu, float* vt, int* ldvt,
                float* work, int* lwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, float* a, int lda );

/* Parameters */
#define M 6
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        float wkopt;
        float* work;
        /* Local arrays */
   /* iwork dimension should be at least 8*min(m,n) */
   int iwork[8*N];
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*N] = {
            7.52f, -0.76f,  5.13f, -4.75f,  1.33f, -2.40f,
           -1.10f,  0.62f,  6.62f,  8.52f,  4.91f, -6.77f,
           -7.95f,  9.34f, -5.66f,  5.75f, -5.49f,  2.34f,
            1.08f, -7.10f,  0.87f,  5.30f, -3.52f,  3.95f
        };
        /* Executable statements */
        printf( " SGESDD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        sgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt,
         &lwork, iwork, &info );
        lwork = (int)wkopt;
        work = (float*)malloc( lwork*sizeof(float) );
        /* Compute SVD */
        sgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work,
```

```
         &lwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of SGESDD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, float* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_sgesdd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_sgesdd Example.
   ======================

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
```

```
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_sgesdd (column-major, high-level) Example Program Results

 Singular values
  18.37  13.63  10.85   4.49

 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
```

```
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*N] = {
            7.52f, -0.76f,  5.13f, -4.75f,  1.33f, -2.40f,
           -1.10f,  0.62f,  6.62f,  8.52f,  4.91f, -6.77f,
           -7.95f,  9.34f, -5.66f,  5.75f, -5.49f,  2.34f,
            1.08f, -7.10f,  0.87f,  5.30f, -3.52f,  3.95f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesdd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_sgesdd( LAPACK_COL_MAJOR, 'S', m, n, a, lda, s,
                        u, ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_sgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_sgesdd Example Program in C for Column Major Data Layout


```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
```

```
*/
/*
   LAPACKE_sgesdd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular matrix A using a divide and conquer method, where A is:

     7.52  -1.10  -7.95   1.08
    -0.76   0.62   9.34  -7.10
     5.13   6.62  -5.66   0.87
    -4.75   8.52   5.75   5.30
     1.33   4.91  -5.49  -3.52
    -2.40  -6.77   2.34   3.95

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a real
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VT

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m orthogonal matrix and VT (V transposed)
   is an n-by-n orthogonal matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VT, not V.

   Example Program Results.
   ========================

 LAPACKE_sgesdd (row-major, high-level) Example Program Results

 Singular values
  18.37  13.63  10.85   4.49

 Left singular vectors (stored columnwise)
  -0.57   0.18   0.01   0.53
   0.46  -0.11  -0.72   0.42
  -0.45  -0.41   0.00   0.36
   0.33  -0.69   0.49   0.19
  -0.32  -0.31  -0.28  -0.61
   0.21   0.46   0.39   0.09

 Right singular vectors (stored rowwise)
  -0.52  -0.12   0.85  -0.03
   0.08  -0.99  -0.09  -0.01
  -0.28  -0.02  -0.14   0.95
   0.81   0.01   0.50   0.31
*/
#include <stdlib.h>
#include <stdio.h>
```

```
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda );

/* Parameters */
#define M 6
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        float s[N], u[LDU*M], vt[LDVT*N];
        float a[LDA*M] = {
            7.52f, -1.10f, -7.95f, 1.08f,
           -0.76f,  0.62f,  9.34f, -7.10f,
            5.13f,  6.62f, -5.66f, 0.87f,
           -4.75f,  8.52f,  5.75f, 5.30f,
            1.33f,  4.91f, -5.49f, -3.52f,
           -2.40f, -6.77f,  2.34f, 3.95f
        };
        /* Executable statements */
        printf( "LAPACKE_sgesdd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_sgesdd( LAPACK_ROW_MAJOR, 'S', m, n, a, lda, s,
                        u, ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_matrix( "Singular values", 1, n, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, n, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", n, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_sgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, float* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

ZGESDD Example

NOTE
For a detailed description and reference information on this function, please visit:

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

The routine computes the singular value decomposition (SVD) of a rectangular complex matrix *A*, optionally the left and/or right singular vectors. This routine uses a divide and conquer algorithm to compute the SVD.

The SVD is written as:

$$A = U*SIGMA*V^H$$

where

- *A* is a complex *m*-by-*n* matrix.
- *SIGMA* is an *m*-by-*n* matrix which is zero except for its $\min(m,n)$ diagonal elements.
- *U* is an *m*-by-*m* unitary matrix.
- V^H (*V* conjugate transposed) is an *n*-by-*n* unitary matrix.

The diagonal elements of *SIGMA* are the singular values of *A*; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of *U* and *V* are the left and right singular vectors of *A*.

The routine returns V^H, not *V*.

Fortran-77 Interface

- Example program in Fortran
- Example program in C

C Interface

Please refer to the Matrix Layout section of the C Interface Conventions for the description of Row Major (C default) and Column Major (Fortran default).

- Example program in C for Row Major Data Layout
- Example program in C for Column Major Data Layout

See Also

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - C

?gesdd function in Intel® oneAPI Math Kernel Library Developer Reference - Fortran

ZGESDD Example Program in Fortran

```
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*  =====================================================================
```

```
*
*  ZGESDD Example.
*  ==============
*
*  Program computes the singular value decomposition of a general
*  rectangular complex matrix A using a divide and conquer method, where A is:
*
*  ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
*  (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
*  (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)
*
*  Description.
*  ============
*
*  The routine computes the singular value decomposition (SVD) of a complex
*  m-by-n matrix A, optionally computing the left and/or right singular
*  vectors. If singular vectors are desired, it uses a divide and conquer
*  algorithm. The SVD is written as
*
*  A = U*SIGMA*VH
*
*  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
*  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
*  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
*  are the singular values of A; they are real and non-negative, and are
*  returned in descending order. The first min(m, n) columns of U and V are
*  the left and right singular vectors of A.
*
*  Note that the routine returns VH, not V.
*
*  Example Program Results.
*  ========================
*
* ZGESDD Example Program Results
*
* Singular values
*  21.76  16.60   3.97
*
* Left singular vectors (stored columnwise)
* (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
* ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
* (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)
*
* Right singular vectors (stored rowwise)
* (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
* ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
* (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*  =====================================================================
*
*     .. Parameters ..
      INTEGER          M, N
      PARAMETER        ( M = 3, N = 4 )
      INTEGER          LDA, LDU, LDVT
      PARAMETER        ( LDA = M, LDU = M, LDVT = N )
      INTEGER          LWMAX
      PARAMETER        ( LWMAX = 1000 )
*
*     .. Local Scalars ..
```

```
      INTEGER          INFO, LWORK
*
*     .. Local Arrays ..
*     IWORK dimension should be at least 8*MIN(M,N)
      INTEGER          IWORK( 8*M )
*     RWORK dimension should be at least 5*(MIN(M,N))**2 + 7*MIN(M,N))
      DOUBLE PRECISION S( M ), RWORK( 5*M*M + 7*M )
      COMPLEX*16       A( LDA, N ), U( LDU, M ), VT( LDVT, N ),
     $                 WORK( LWMAX )
      DATA             A/
     $ (-5.40, 7.40),( 1.09, 1.55),( 9.88, 1.91),
     $ ( 6.00, 6.38),( 2.60, 0.07),( 4.92, 6.31),
     $ ( 9.91, 0.16),( 3.98,-5.26),(-2.11, 7.39),
     $ (-5.28,-4.16),( 2.03, 1.11),(-9.81,-8.98)
     $                 /
*
*     .. External Subroutines ..
      EXTERNAL         ZGESDD
      EXTERNAL         PRINT_MATRIX, PRINT_RMATRIX
*
*     .. Intrinsic Functions ..
      INTRINSIC        INT, MIN
*
*     .. Executable Statements ..
      WRITE(*,*)'ZGESDD Example Program Results'
*
*     Query the optimal workspace.
*
      LWORK = -1
      CALL ZGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, RWORK, IWORK, INFO )
      LWORK = MIN( LWMAX, INT( WORK( 1 ) ) )
*
*     Compute SVD.
*
      CALL ZGESDD( 'Singular vectors', M, N, A, LDA, S, U, LDU, VT,
     $             LDVT, WORK, LWORK, RWORK, IWORK, INFO )
*
*     Check for convergence.
*
      IF( INFO.GT.0 ) THEN
         WRITE(*,*)'The algorithm computing SVD failed to converge.'
         STOP
      END IF
*
*     Print singular values.
*
      CALL PRINT_RMATRIX( 'Singular values', 1, M, S, 1 )
*
*     Print left singular vectors.
*
      CALL PRINT_MATRIX( 'Left singular vectors (stored columnwise)',
     $                   M, M, U, LDU )
*
*     Print right singular vectors.
*
      CALL PRINT_MATRIX( 'Right singular vectors (stored rowwise)',
     $                   M, N, VT, LDVT )
```

```
      STOP
      END
*
*     End of ZGESDD Example.
*
*  =============================================================================
*
*     Auxiliary routine: printing a matrix.
*
      SUBROUTINE PRINT_MATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      COMPLEX*16       A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,'(',F6.2,',',F6.2,')') )
      RETURN
      END
*
*     Auxiliary routine: printing a real matrix.
*
      SUBROUTINE PRINT_RMATRIX( DESC, M, N, A, LDA )
      CHARACTER*(*)    DESC
      INTEGER          M, N, LDA
      DOUBLE PRECISION A( LDA, * )
*
      INTEGER          I, J
*
      WRITE(*,*)
      WRITE(*,*) DESC
      DO I = 1, M
         WRITE(*,9998) ( A( I, J ), J = 1, N )
      END DO
*
 9998 FORMAT( 11(:,1X,F6.2) )
      RETURN
      END
```

ZGESDD Example Program in C

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
```

```
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   ZGESDD Example.
   ==============

   Program computes the singular value decomposition of a general
   rectangular complex matrix A using a divide and conquer method, where A is:

   ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
   (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
   (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 ZGESDD Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>

/* Complex datatype */
```

```
struct _dcomplex { double re, im; };
typedef struct _dcomplex dcomplex;

/* ZGESDD prototype */
extern void zgesdd( char* jobz, int* m, int* n, dcomplex* a,
                int* lda, double* s, dcomplex* u, int* ldu, dcomplex* vt, int* ldvt,
                dcomplex* work, int* lwork, double* rwork, int* iwork, int* info );
/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, int m, int n, dcomplex* a, int lda );
extern void print_rmatrix( char* desc, int m, int n, double* a, int lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        int m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info, lwork;
        dcomplex wkopt;
        dcomplex* work;
        /* Local arrays */
        /* iwork dimension should be at least 8*min(m,n) */
        int iwork[8*M];
        /* rwork dimension should be at least 5*(min(m,n))**2 + 7*min(m,n)) */
        double s[M], rwork[5*M*M+7*M];
        dcomplex u[LDU*M], vt[LDVT*N];
        dcomplex a[LDA*N] = {
           {-5.40,  7.40}, { 1.09,  1.55}, { 9.88,  1.91},
           { 6.00,  6.38}, { 2.60,  0.07}, { 4.92,  6.31},
           { 9.91,  0.16}, { 3.98, -5.26}, {-2.11,  7.39},
           {-5.28, -4.16}, { 2.03,  1.11}, {-9.81, -8.98}
        };
        /* Executable statements */
        printf( " ZGESDD Example Program Results\n" );
        /* Query and allocate the optimal workspace */
        lwork = -1;
        zgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &wkopt,
         &lwork, rwork, iwork, &info );
        lwork = (int)wkopt.re;
        work = (dcomplex*)malloc( lwork*sizeof(dcomplex) );
        /* Compute SVD */
        zgesdd( "Singular vectors", &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, work,
         &lwork, rwork, iwork, &info );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
```

```
        /* Free workspace */
        free( (void*)work );
        exit( 0 );
} /* End of ZGESDD Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, int m, int n, dcomplex* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].re, a[i+j*lda].im );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, int m, int n, double* a, int lda ) {
        int i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```

LAPACKE_zgesdd Example Program in C for Row Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesdd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A using a divide and conquer method, where A is:

   ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
   (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
   (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

   Description.
   ============
```

```
  The routine computes the singular value decomposition (SVD) of a complex
  m-by-n matrix A, optionally computing the left and/or right singular
  vectors. If singular vectors are desired, it uses a divide and conquer
  algorithm. The SVD is written as

  A = U*SIGMA*VH

  where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
  diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
  transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
  are the singular values of A; they are real and non-negative, and are
  returned in descending order. The first min(m, n) columns of U and V are
  the left and right singular vectors of A.

  Note that the routine returns VH, not V.

  Example Program Results.
  ========================

 LAPACKE_zgesdd (row-major, high-level) Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA N
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[M];
        MKL_Complex16 u[LDU*M], vt[LDVT*N];
        MKL_Complex16 a[LDA*M] = {
```

```
           {-5.40,  7.40}, { 6.00,  6.38}, { 9.91,  0.16}, {-5.28, -4.16},
           { 1.09,  1.55}, { 2.60,  0.07}, { 3.98, -5.26}, { 2.03,  1.11},
           { 9.88,  1.91}, { 4.92,  6.31}, {-2.11,  7.39}, {-9.81, -8.98}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesdd (row-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_zgesdd( LAPACK_ROW_MAJOR, 'S', m, n, a, lda, s, u,
         ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_zgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i*lda+j].real, a[i*lda+j].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i*lda+j] );
                printf( "\n" );
        }
}
```

LAPACKE_zgesdd Example Program in C for Column Major Data Layout

```
/*****************************************************************************
*  Copyright (C) 2009-2015 Intel Corporation. All Rights Reserved.
*  The information and material ("Material") provided below is owned by Intel
*  Corporation or its suppliers or licensors, and title to such Material remains
*  with Intel Corporation or its suppliers or licensors. The Material contains
*  proprietary information of Intel or its suppliers and licensors. The Material
*  is protected by worldwide copyright laws and treaty provisions. No part of
*  the Material may be copied, reproduced, published, uploaded, posted,
*  transmitted, or distributed in any way without Intel's prior express written
*  permission. No license under any patent, copyright or other intellectual
*  property rights in the Material is granted to or conferred upon you, either
*  expressly, by implication, inducement, estoppel or otherwise. Any license
```

```
*  under such intellectual property rights must be express and approved by Intel
*  in writing.
*
*****************************************************************************
*/
/*
   LAPACKE_zgesdd Example.
   =======================

   Program computes the singular value decomposition of a general
   rectangular complex matrix A using a divide and conquer method, where A is:

   ( -5.40,  7.40) (  6.00,  6.38) (  9.91,  0.16) ( -5.28, -4.16)
   (  1.09,  1.55) (  2.60,  0.07) (  3.98, -5.26) (  2.03,  1.11)
   (  9.88,  1.91) (  4.92,  6.31) ( -2.11,  7.39) ( -9.81, -8.98)

   Description.
   ============

   The routine computes the singular value decomposition (SVD) of a complex
   m-by-n matrix A, optionally computing the left and/or right singular
   vectors. If singular vectors are desired, it uses a divide and conquer
   algorithm. The SVD is written as

   A = U*SIGMA*VH

   where SIGMA is an m-by-n matrix which is zero except for its min(m,n)
   diagonal elements, U is an m-by-m unitary matrix and VH (V conjugate
   transposed) is an n-by-n unitary matrix. The diagonal elements of SIGMA
   are the singular values of A; they are real and non-negative, and are
   returned in descending order. The first min(m, n) columns of U and V are
   the left and right singular vectors of A.

   Note that the routine returns VH, not V.

   Example Program Results.
   ========================

 LAPACKE_zgesdd (column-major, high-level) Example Program Results

 Singular values
  21.76  16.60   3.97

 Left singular vectors (stored columnwise)
 (  0.55,  0.00) (  0.76,  0.00) ( -0.34,  0.00)
 ( -0.04, -0.15) (  0.27, -0.23) (  0.55, -0.74)
 (  0.81,  0.12) ( -0.52, -0.14) (  0.13, -0.11)

 Right singular vectors (stored rowwise)
 (  0.23,  0.21) (  0.37,  0.39) (  0.24,  0.33) ( -0.56, -0.37)
 ( -0.58,  0.40) (  0.11,  0.17) (  0.60, -0.27) (  0.16,  0.06)
 (  0.60,  0.12) ( -0.19,  0.30) (  0.39,  0.20) (  0.45,  0.31)
*/
#include <stdlib.h>
#include <stdio.h>
#include "mkl_lapacke.h"

/* Auxiliary routines prototypes */
```

```
extern void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda );
extern void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda );

/* Parameters */
#define M 3
#define N 4
#define LDA M
#define LDU M
#define LDVT N

/* Main program */
int main() {
        /* Locals */
        MKL_INT m = M, n = N, lda = LDA, ldu = LDU, ldvt = LDVT, info;
        /* Local arrays */
        double s[M];
        MKL_Complex16 u[LDU*M], vt[LDVT*N];
        MKL_Complex16 a[LDA*N] = {
           {-5.40,  7.40}, { 1.09,  1.55}, { 9.88,  1.91},
           { 6.00,  6.38}, { 2.60,  0.07}, { 4.92,  6.31},
           { 9.91,  0.16}, { 3.98, -5.26}, {-2.11,  7.39},
           {-5.28, -4.16}, { 2.03,  1.11}, {-9.81, -8.98}
        };
        /* Executable statements */
        printf( "LAPACKE_zgesdd (column-major, high-level) Example Program Results\n" );
        /* Compute SVD */
        info = LAPACKE_zgesdd( LAPACK_COL_MAJOR, 'S', m, n, a, lda, s, u,
         ldu, vt, ldvt );
        /* Check for convergence */
        if( info > 0 ) {
                printf( "The algorithm computing SVD failed to converge.\n" );
                exit( 1 );
        }
        /* Print singular values */
        print_rmatrix( "Singular values", 1, m, s, 1 );
        /* Print left singular vectors */
        print_matrix( "Left singular vectors (stored columnwise)", m, m, u, ldu );
        /* Print right singular vectors */
        print_matrix( "Right singular vectors (stored rowwise)", m, n, vt, ldvt );
        exit( 0 );
} /* End of LAPACKE_zgesdd Example */

/* Auxiliary routine: printing a matrix */
void print_matrix( char* desc, MKL_INT m, MKL_INT n, MKL_Complex16* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
                for( j = 0; j < n; j++ )
                        printf( " (%6.2f,%6.2f)", a[i+j*lda].real, a[i+j*lda].imag );
                printf( "\n" );
        }
}

/* Auxiliary routine: printing a real matrix */
void print_rmatrix( char* desc, MKL_INT m, MKL_INT n, double* a, MKL_INT lda ) {
        MKL_INT i, j;
        printf( "\n %s\n", desc );
        for( i = 0; i < m; i++ ) {
```

```
                for( j = 0; j < n; j++ ) printf( " %6.2f", a[i+j*lda] );
                printf( "\n" );
        }
}
```