LAPACK Working Note 117
A Fortran 90 Interface for LAPACK: LAPACK90, version 1.0

L. Susan Blackford* Jack J. Dongarra† Jeremy Du Croz‡
Sven Hammarling§ Jerzy Waśniewski¶

December 5, 1996

Abstract

The purpose of this report is to discuss the design of a Fortran 90 interface to LAPACK. Our emphasis at this stage is on the design of an improved user-interface to the package, taking advantage of the considerable simplifications which Fortran 90 allows. The proposed design makes use of assumed-shape arrays, optional arguments, and generic interfaces. The Fortran 90 interface can be implemented initially by writing Fortran 90 jackets to call the existing Fortran 77 code, and can persist unchanged even if the underlying Fortran 77 LAPACK code is rewritten to take advantage of the new features of Fortran 90. We aim to maintain a comparable level of performance as with the Fortran 77 code. In this paper we implement interfaces to the subset of LAPACK routines for solving systems of linear equations $AX = B$ with a general matrix A, and for symmetric and Hermitian eigenproblems.

*(formerly L. S. Ostrouchov) Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA; email: susan@cs.utk.edu

†Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA and Mathematical Sciences Section, Oak Ridge National Laboratory, P.O.Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367, USA; email: dongarra@cs.utk.edu

‡Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK; email: jeremy@nag.co.uk

§Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK; email: sven@nag.co.uk

¶UNI•C, The Danish Computing Centre for Research and Education, DTU, Bldg. 304, DK-2800 Lyngby, Denmark; Email: jerzy.wasniewski@uni-c.dk

Contents

1 Introduction

This paper is a follow on paper to LAPACK WN 101 [2] which proposed an interface to some of the LAPACK [1] linear equation routines. Following comments to that paper we have added some additional functionality to the interface. In this paper we also consider the symmetric eigenproblem. We welcome comments on the proposal given here. Our emphasis at this stage is on the design of an improved *user-interface* to the package, taking advantage of the considerable simplifications which Fortran 90 allows (see [3]).

The Fortran 90 interface can be implemented initially by writing Fortran 90 jackets to call the existing Fortran 77 code, and can persist unchanged even if the underlying Fortran 77 LAPACK code is rewritten to take advantage of the new features of Fortran 90.

Although we would like to maintain a comparable level of performance to the Fortran 77 LAPACK code, due to the immaturity of many of the current Fortran 90 compilers, we cannot usually at present achieve this same level of performance. We reiterate that our goal is to provide a true Fortran 90 interface to LAPACK. We are aware of certain modifications to the interface which could enhance performance, but these modifications complicate the interface by requiring Fortran 77-ish constructs. If performance is the main focus behind the user's application, we recommend that the user call the Fortran 77 interface directly.

For convenience we use the name "LAPACK 77" to denote the existing Fortran 77 package, and "LAPACK 90" to denote the new Fortran 90 interface which we are proposing.

2 LAPACK 77 and Fortran 90 Compilers

2.1 Linking LAPACK 77 to Fortran 90 programs

LAPACK 77 can be called from Fortran 90 programs in its present form — with some qualifications. The qualifications arise only because LAPACK 77 is not written entirely in *standard* Fortran 77; the exceptions are the use of the `COMPLEX*16` data type and related intrinsic functions, as listed in Section 6.1 of [1]; these facilities are provided as extensions to the standard language by many Fortran 77 and Fortran 90 compilers. Equivalent facilities are provided in standard Fortran 90, using the parameterized form of the `COMPLEX` data type (see below).

To link LAPACK 77 to a Fortran 90 program (which must of course be compiled by a Fortran 90 compiler), one of the following approaches will be necessary, depending on the compilers available.

1. Link the Fortran 90 program to an existing LAPACK 77 library, compiled by a Fortran 77 compiler. This approach can only work if the compilers

have been designed to allow cross-linking.

2. If such cross-linking is not possible, recompile and archive the LAPACK 77 library with the Fortran 90 compiler, provided that the compiler accepts `COMPLEX*16` and related intrinsics as extensions.

3. If these extensions are not accepted and the user requires this data type, the LAPACK 77 code must be rewritten in standard Fortran 90 (see below).

   Some conversions needed to use the double precision complex data type in standard Fortran 90 code from LAPACK 77 are:

`COMPLEX*16`	⇒	`COMPLEX(KIND=Kind(0.0D0)`
`DCONJG(z)` for `COMPLEX*16 z`	⇒	`CONJG(z)`
`DBLE(z)` for `COMPLEX*16 z`	⇒	`REAL(z)`
`DIMAG(z)` for `COMPLEX*16 z`	⇒	`AIMAG(z)`
`DCMPLX(x,y)` for `DOUBLE PRECISION x, y`	⇒	`CMPLX(x,y,KIND=Kind(0.0D0))`

One further obstacle may remain: it is possible that if LAPACK 77 has been recompiled with a Fortran 90 compiler, it may not link correctly to an optimized assembly-language BLAS library that has been designed to interface with Fortran 77. Until this is rectified by the vendor of the BLAS library, Fortran 77 code for the BLAS must be used.

2.2 Interface blocks for LAPACK 77

Fortran 90 allows one immediate extra benefit to be provided to Fortran 90 users of LAPACK 77, without making any further changes to the existing code: that is a *module* of *explicit interfaces* for the routines. If this module is accessed by a `USE` statement in any program unit which makes calls to LAPACK routines, then those calls can be checked by the compiler for errors in the numbers or types of arguments.

The module can be constructed by extracting the necessary specification statements from the Fortran 77 code with a little modification, as illustrated by the following example containing an interface for the single routine `SSYEV`:

```
MODULE LA_SF77MOD
   INTERFACE
      SUBROUTINE SSYEV( JOBZ, UPLO, N, A, LDA, W, &
                        WORK, LWORK, INFO )
         USE LA_PRECISION, ONLY: WP => SP
         CHARACTER(LEN=1), INTENT(IN) :: JOBZ, UPLO
         INTEGER, INTENT(IN) :: LDA, LWORK, N
         INTEGER, INTENT(OUT) :: INFO
         REAL(WP), INTENT(INOUT) :: A(LDA,*)
         REAL(WP), INTENT(OUT) :: W(*), WORK(*)
      END SUBROUTINE SSYEV
                 ...   ...   ...
   END INTERFACE
                 ...   ...   ...
END MODULE LA_SF77MOD
```

A module containing interfaces for all of the routines in LAPACK 77 will be required; here we provide one module per precision, for example LA_SF77MOD, LA_DF77MOD, LA_CF77MOD, and LA_ZF77MOD, and one module for auxiliary routines, LA_AUXMOD.

3 Proposed Design of the LAPACK 90 interface

In the design of a Fortran 90 interface to LAPACK, we propose to take advantage of the features of the language listed below.

1. **Assumed-shape arrays:** All array arguments to LAPACK 90 routines will be assumed-shape arrays. Arguments to specify problem dimensions or array dimensions will not be required.

   This implies that the actual arguments supplied to LAPACK routines *must* have the *exact* shape required by the problem. The most convenient ways to achieve this are:

   - using allocatable arrays, for example:

     ```
     REAL, ALLOCATABLE :: A(:,:), B(:)
      . . .
     ALLOCATE( A(N,N), B(N) )
      . . .
     CALL LA_GESV( A, B )
     ```

   - passing array sections, for example:

```
REAL :: A(NMAX,NMAX), B(NMAX)
 . . .
CALL LA_GESV( A(:N,:N), B(:N) )
```

Zero dimensions (empty arrays) will be allowed.

There are some grounds for concern about the effect of Fortran 77 assumed-size arrays on performance because compilers cannot assume that their storage is contiguous. The effect on performance will of course depend on the compiler, and may diminish in time as compilers become more effective in optimizing compiled code. See section 7.

2. **Automatic allocation of work arrays:** Workspace arguments and arguments to specify their dimensions will not be needed. In simple cases, *automatic arrays* of the required size can be declared internally. In other cases, allocatable arrays may need to be declared and explicitly allocated. Explicit allocation is needed in particular when the amount of workspace required depends on the block-size to be used (which is not passed as an argument).

3. **Optional arguments:** In LAPACK 77, character arguments are frequently used to specify some choice of options. In Fortran 90, a choice of options can sometimes be specified naturally by the presence or absence of optional arguments: for example, options to compute the left or right eigenvectors can be specified by the presence of arguments `VL` or `VR`, and the character arguments `JOBVL` and `JOBVR` which are required in the LAPACK 77 routine `DGEEV`, are not needed in LAPACK 90.

   In other routines, a character argument to specify options may still be required, but can itself be made optional if there is a natural default value: for example, in `DGESVX` the argument `TRANS` can be made optional, with default value `'N'`.

   Optional arguments can also help to combine two or more routines into one: for example, the functionality provided by the routine `DGECON` can be made accessible by adding an optional argument `RCOND` to `DGETRF`.

4. **Generic Interfaces:** The systematic occurrence in LAPACK of analogous routines for real or complex data, and for single or double precision lends itself well to the definition of generic interfaces, allowing four different routines to be accessed through the same generic name.

   Generic interfaces can also be used to cover routines whose arguments differ in *rank*, and thus provide a slight increase in flexibility over LAPACK 77. For example, in LAPACK 77, routines for solving a system of linear equations (such as DGESV), allow for multiple right hand sides, and so the arrays which hold the right hand sides and solutions are always of rank 2. In LAPACK 90, we can provide alternative versions of the routines (covered by a single generic interface) in which the arrays holding the right hand sides and solutions may *either* be of rank 1 (for a single right hand side) *or* be of rank 2 (for several right hand sides).

5. **Naming:** For the generic routine names, we propose:

   - the initial letter (`S`, `C`, `D` or `Z`) is simply omitted.
   - the letters `LA_` are prefixed to all names to identify them as names of LAPACK routines.

   In other respects the naming scheme remains the same as described in Section 2.1.3 of [1]: for example, `LA_GESV`.

   It would also be possible to define longer, more meaningful names (which could co-exist with the shorter names), but we have not attempted this here.

   We have *not* proposed the use of any *derived types* in this Fortran 90 interface. They could be considered — for example, to hold the details of an *LU* factorization and equilibration factors. However, since LAPACK routines are so frequently used as building blocks in larger algorithms or applications, we feel that there are advantages in keeping the interface simple, and avoiding possible loss of efficiency.

6. **Error-handling:**

   In LAPACK 77, all documented routines have a diagnostic output argument `INFO`. Three types of exit from a routine are allowed:

   successful termination: the routine returns to the calling program with `INFO` set to 0.

   illegal value of one or more arguments, or error in store allocation: the routine sets `INFO` < 0 and calls the auxiliary routine `XERBLA`; the standard version of `XERBLA` issues an error message identifying the first invalid argument, and stops execution.

   failure in the course of computation: the routine sets `INFO` > 0 and returns to the calling program without issuing any error message. Only some LAPACK 77 routines need to allow this type of error-exit; it is then the responsibility of a user to test `INFO` on return to the calling program.

   For LAPACK 90 we propose that the argument `INFO` becomes *optional*: if it is not present and an error occurs, then the routine *always* issues an error message and stops execution, even when `INFO` > 0 (in which case the error message reports the value of `INFO`). If a user wishes to continue execution after a failure in computation, then `INFO` must be supplied and tested on return.

This behaviour simplifies calls to LAPACK 90 routines when there is no need to test `INFO` on return, and makes it less likely that users will forget to test `INFO` when necessary.

If an invalid argument is detected, we propose that routines issue an error message and stop, as in LAPACK 77. Note however that in Fortran 90 there can be different reasons for an argument being invalid:

illegal value : as in LAPACK 77.

invalid shape (of an assumed-shape array): for example, a 2-dimensional array is not square when it is required to be.

inconsistent shapes (of two or more assumed-shape arrays): for example, arrays holding the right hand sides and solutions of a system of linear equations must have the same shape.

No more core allocation needed for the LAPACK 77.

The specification could be extended so that the error-message could distinguish between these cases. For more detail see in appendix section D.2.

4 Prototype LAPACK 90 Interfaces

We have implemented Fortran 90 jacket procedures to the subset of LAPACK 77 routines concerned with the solution of systems of linear equations $AX = B$ for a general matrix A — that is, the driver routines `xGESV` and `xGESVX`, and the computational routines `xGETRF`, `xGETRS`, `xGETRI`, `xGECON`, `xGERFS` and `xGEEQU`. We also consider here the symmetric and Hermitian eigenproblem routines `xSYTRD`, `xSYGV`, `xSYGST`, `xORGTR`, `xSYEV`, `xSYEVD`, `xSYEVX` and the factor routine `xPOTRF` which is strongly connected with the `xSYGST` subroutines.

Here we present calling sequences for each of the proposed routines, the first without using any of the optional arguments, the second using all the arguments. For ease of comparison between LAPACK 77 and LAPACK 90, we have retained the same names for the corresponding arguments, although of course Fortran 90 offers the possibility of longer names (for example, `IPIV` could become `PIVOT_INDICES`). In this prototype implementation, we have assumed that the code of LAPACK 77 is not modified.

Detailed documentation of the proposed interfaces can be found in Appendices A, B and C.

4.1 Solution of Systems of Linear Equations for a General Matrix A

`LA_GESV` (simple driver):

```
CALL LA_GESV( A, B )

CALL LA_GESV( A, B, IPIV, INFO )
```

Comments:

- The array `B` may have rank 1 (one right hand side) or rank 2 (several right hand sides).

`LA_GESVX` (expert driver):

```
CALL LA_GESVX( A, B, X )

CALL LA_GESVX( A, B, X, AF, IPIV, FACT, TRANS, EQUED, R, C, &
               FERR, BERR, RCOND, RPVGRW, INFO )
```

Comments:

- The arrays `B` and `X` may have rank 1 (in which case `FERR` and `BERR` are scalars) or rank 2 (in which case `FERR` and `BERR` are rank-1 arrays).
- `RPVGRW` returns the reciprocal pivot growth factor (returned in `WORK(1)` in LAPACK 77).
- the presence or absence of `EQUED` is used to specify whether or not equilibration is to be performed, instead of the option `FACT = 'E'`.

`LA_GETRF` (LU factorization):

```
CALL LA_GETRF( A, IPIV )

CALL LA_GETRF( A, IPIV, RCOND, NORM, INFO )
```

Comments:

- instead of a separate routine `LA_GECON`, we propose that optional arguments `RCOND` and `NORM` are added to `LA_GETRF` to provide the same functionality in a more convenient manner. The argument `ANORM` of `xGECON` is not needed, because `LA_GETRF` can always compute the norm of A if required.

`LA_GETRS` (solution of equations using LU factorization):

```
CALL LA_GETRS( A, IPIV, B )

CALL LA_GETRS( A, IPIV, B, TRANS, INFO )
```

Comments:

- The array B may have rank 1 or 2.

LA_GETRI (matrix inversion using *LU* factorization):

```
CALL LA_GETRI( A, IPIV )

CALL LA_GETRI( A, IPIV, INFO )
```

LA_GERFS (refine solution of equations and optionally compute error bounds):

```
CALL LA_GERFS( A, AF, IPIV, B, X )

CALL LA_GERFS( A, AF, IPIV, B, X, TRANS, FERR, BERR, INFO )
```

Comments:

- The arrays B and X may have rank 1 (in which case FERR and BERR are scalars) or rank 2 (in which case FERR and BERR are rank-1 arrays).

LA_GEEQU (equilibration):

```
CALL LA_GEEQU( A, R, C )

CALL LA_GEEQU( A, R, C, ROWCND, COLCND, AMAX, INFO )
```

4.2 Symmetric and Hermitian Eigenproblem Routines

LA_SYEV / LA_HEEV (all eigenvalues and optionally eigenvectors):

```
CALL LA_SYEV / LA_HEEV( A, W )

CALL LA_SYEV / LA_HEEV( A, W, JOBZ, UPLO, INFO )
```

LA_SYEVD / LA_HEEVD (all eigenvalues and optionally eigenvectors using a divide and conquer algorithm):

```
CALL LA_SYEVD / LA_HEEVD( A, W )

CALL LA_SYEVD / LA_HEEVD( A, W, JOBZ, UPLO, INFO )
```

LA_SYEVX / LA_HEEVX (selected eigenvalues and optionally eigenvectors):

```
CALL LA_SYEVX / LA_HEEVD( A, W )

CALL LA_SYEVX / LA_HEEVD( A, W, JOBZ, UPLO, VL, VU, IL, IU, &
                          M, IFAIL, ABSTOL, INFO )
```

LA_SYGV / LA_HEGV (all eigenvalues and optionally eigenvectors of the form $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$

```
CALL LA_SYGV / LA_HEGV( A, B, W )

CALL LA_SYGV / LA_HEGV( A, B, W, ITYPE, JOBZ, UPLO, INFO )
```

LA_SYGST / LA_HEGST (reduction to standard form):

```
CALL LA_SYGST / LA_HEGST( A, B )

CALL LA_SYGST / LA_HEGST( A, B, ITYPE, UPLO, INFO )
```

LA_SYTRD / LA_HETRD (reduction to tridiagonal form):

```
CALL LA_SYTRD / LA_HETRD( A, TAU )

CALL LA_SYTRD / LA_HETRD( A, TAU, UPLO, INFO )
```

LA_ORGTR / LA_UNGTR (generates an orthogonal matrix):

```
CALL LA_ORGTR / LA_UNGTR( A, TAU )

CALL LA_ORGTR / LA_UNGTR( A, TAU, UPLO, INFO )
```

LA_POTRF (generates an orthogonal matrix):

```
CALL LA_POTRF( A )

CALL LA_POTRF( A, UPLO, RCOND, NORM, INFO )
```

5 Documentation

In appendices A, B and C, we give a first attempt at draft documentation for these routines. The style is somewhat similar to that of the LAPACK Users' Guide, but with various obvious new conventions introduced to handle the generic nature of the interfaces.

6 Test Software

Additional test software is being developed to test the new interfaces. At present, the test software is a modified version of the LAPACK 77 Test Suite.

7 Performance Issues and Timings

The present FORTRAN 90 compilers do not assume that array storage is contiguous. The effect on performance will of course depend on their compiler. The contents of arrays are copied to the temporary storage when calling FORTRAN 77 subroutines (functions) from the FORTRAN 90 procedures. This data copy results in a degradation in performance. As previously mentioned, we are aware of performance enhancements to the interface which could be made available through alternative interfaces to the routines. However, these enhancements violate our goal to present a true Fortran 90 interface to the LAPACK library by complicating the design of the interface through the use of Fortran 77 constructs. Our Fortran 90 interface is provided as a simplified user interface to the LAPACK library. If performance is the main focus of the user's application, he should call the Fortran 77 interface directly.

We have performed timings to measure the extra overhead of the Fortran 90 interface. We timed `LA_GETRF` on a single processor of an IBM SP-2 (in double precision) and a single processor of a Cray YMP C90A (in single precision). All timings are given in megaflops.

IBM
1. Speed of LAPACK 90 calling LAPACK 77 and BLAS from the ESSL library.
2. Speed of LAPACK 77, using BLAS from the ESSL library.

Array size	600	700	800	900	1000	1100	1200	1300	1400	1500
LAPACK90	187	180	182	170	172	172	176	177	181	182
LAPACK77	191	181	182	171	172	173	176	179	180	182

Cray
1. Speed of LAPACK 90 calling LAPACK 77 as provided by CRAY in LIBSCI.
2. Speed of LAPACK 77 as provided by CRAY in LIBSCI.

Array size	600	700	800	900	1000	1100	1200	1300	1400	1500
LAPACK90	723	828	646	841	822	855	789	857	846	868
LAPACK77	778	834	649	845	825	860	794	864	848	873

The above tables show the LAPACK 90 results are a little slower (1 or 2%) than the LAPACK 77 results.

8 Acknowledgments

This research was partially supported by the Danish Natural Science Research Council through a grant for the EPOS project (Efficient Parallel Algorithms for Optimization and Simulation).

References

[1] E. Anderson, Z. Bai, C. H. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. C. Sorensen. *LAPACK Users' Guide Release 2.0.* SIAM, Philadelphia, 1995.

[2] J.J. Dongarra, J. Du Croz, S. Hammarling, J. Waśniewski and A. Zemła. *LAPACK Working Note 101, A Proposal for a Fortran 90 Interface for LAPACK.* Report UNIC-95-9, UNI•C, Lyngby, Denmark, 1995. Report ut-cs-95-295, University of Tennessee, Computer Science Department, Knoxville, July, 1995.

[3] M. Metcalf and J. Reid. *Fortran 90 Explained.* Oxford, New York, Tokyo, Oxford University Press, 1990.

A Solving Systems of Linear Equations $AX = B$ with a General Matrix A, Documentation

A.1 LA_GESV

A.1.1 Purpose

LA_GESV computes the solution to either a real or complex system of linear equations $AX = B$, where A is a square matrix and B and X are either rectangular matrices or vectors.

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

A.1.2 Specification

SUBROUTINE LA_GESV(A, B, IPIV, INFO)
type(*wp*), INTENT(INOUT) :: A(:,:), *rhs*
INTEGER, INTENT(OUT), OPTIONAL :: IPIV(:)
INTEGER, INTENT(OUT), OPTIONAL :: INFO
where
type ::= REAL | COMPLEX
wp ::= KIND(1.0) | KIND(1.0D0)
rhs ::= B(:,:) | B(:)

A.1.3 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$, $size(A,1) = size(A,2)$.

- On entry, the matrix A.
- On exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.

B – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$ or $(:)$, and $size(B,1)$ or $size(B) = size(A,1)$.

- On entry, the right hand side vector(s) of matrix B for the system of equations $AX = B$.
- On exit, if there is no error, the matrix of solution vector(s) X.

IPIV – *Optional* (*output*) **INTEGER** array, shape $(:)$, $size(IPIV) = size(A,1)$.

- If $IPIV$ is present, it contains indices that define the permutation matrix P; row i of the matrix was interchanged with row $IPIV(i)$.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  $= 0$: successful exit
  < 0 : if $INFO = -i$, the i-th argument had an illegal value
  ≥ 0 : if $INFO = i$, $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.
- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.2 LA_GESVX

A.2.1 Purpose

LA_GESVX computes the solution to a real or complex system of linear equations $AX = B$, where A is a square matrix and B and X are either rectangular matrices or vectors.

LA_GESVX is an expert driver routine, which can also optionally perform the following functions:

- solve $A^T X = B$ or $A^H X = B$,
- estimate the condition number of A
- return the pivot growth factor
- refine the solution and compute forward and backward error bounds
- equilibrate the system if A is poorly scaled.

A.2.2 Specification

SUBROUTINE LA_GESVX (A, B, X, AF, IPIV, FACT, TRANS, EQUED, &
R, C, FERR, BERR, RCOND, RPVGRW, INFO)
type(*wp*), INTENT(INOUT) :: A(:,:), *rhs*
type(*wp*), INTENT(OUT) :: *sol*
type(*wp*), INTENT(INOUT), OPTIONAL :: AF(:,:)
INTEGER, INTENT(INOUT), OPTIONAL :: IPIV(:)
CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: FACT, TRANS
CHARACTER(LEN=1), INTENT(INOUT), OPTIONAL :: EQUED
REAL(*wp*), INTENT(INOUT), OPTIONAL :: R(:), C(:)
REAL(*wp*), INTENT(OUT), OPTIONAL :: *err*, RCOND, RPVGRW

INTEGER, INTENT(OUT), OPTIONAL :: INFO
where
type ::= REAL | COMPLEX
wp ::= KIND(1.0) | KIND(1.0D0)
rhs ::= B(:,:) | B(:)
sol ::= X(:,:) | X(:)
err ::= FERR(:), BERR(:) | FERR, BERR

A.2.3 Description

The following steps are performed:

1. If $FACT$ is not present or $FACT = 'N'$, and $EQUED$ is present, real scaling factors are computed to equilibrate the system:

   TRANS = 'N' : $diag(R)\ A\ diag(C)\ (diag(C))^{-1}\ X = diag(R)\ B$
   TRANS = 'T' : $(diag(R)\ A\ diag(C))^T\ (diag(R))^{-1}\ X = diag(C)\ B$
   TRANS = 'C' : $(diag(R)\ A\ diag(C))^H\ (diag(R))^{-1}\ X = diag(C)\ B$

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by $diag(R)\ A\ diag(C)$ and B by $diag(R)\ B$ (if $TRANS = 'N'$) or $diag(C)\ B$ (if $TRANS = 'T'$ or $'C'$).

2. If $FACT = 'N'$, the LU decomposition is used to factor the matrix A (after equilibration if $EQUED$ is present) as $A = PLU$, where P is a permutation matrix, L is a unit lower triangular matrix, and U is upper triangular.

3. The factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, steps $4 - 6$ are skipped.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by $diag(C)$ (if $TRANS = 'N'$) or $diag(R)$ (if $TRANS = 'T'$ or $'C'$) so that it solves the original system before equilibration.

A.2.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$, $size(A,1) = size(A,2)$.

If $FACT$ is not present or $FACT = 'N'$,

- On entry, the matrix A.
- On exit, if $EQUED$ is present, the matrix A may have been overwritten by the equilibrated matrix (see $EQUED$).

If $FACT$ is present and $FACT = 'F'$,

- On entry, the matrix A, possibly equilibrated in a previous call to **LA_GESVX** (see $EQUED$).
- On exit, A is unchanged.

B – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$ or $(:)$, and $size(B,1)$ or $size(B) = size(A,1)$.

- On entry, the right hand side vector(s) of matrix B for the system of equations $AX = B$.
- On exit, if $EQUED$ is present, B may have been scaled in accordance with the equilibration of A (see $EQUED$); otherwise, B is unchanged.

X – (*output*) **REAL** or **COMPLEX** array, shape $(:,:)$ or $(:)$, $size(X,1)$ or $size(X) = size(A,1)$.
If $INFO = 0$, the solution matrix (vector) X to the original system of equations. Note that X always returns the solution to the *original* system of equations; if equilibration has been performed ($EQUED$ is present and $EQUED \neq 'N'$), this does not correspond to the scaled A and B.

AF – *Optional* (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$, $size(AF,1) = size(AF,2) = size(A,1)$.

If $FACT$ is not present or $FACT = 'N'$, then AF is an *output* argument and returns the factors L and U from the factorization $A = PLU$ of the original matrix A, possibly equilibrated if $EQUED$ is present.

If $FACT$ is present and $FACT = 'F'$, then AF is an *input* argument (and must be present); on entry, it must contain the factors L and U of A (possibly equilibrated if $EQUED$ is present), returned by a previous call to **LA_GESVX**.

IPIV – *Optional* (*input/output*) **INTEGER** array, shape $(:)$, $size(IPIV) = size(A,1)$.

If $FACT$ is not present or $FACT = 'N'$, then $IPIV$ is an *output* argument and returns the pivot indices from the factorization $A = PLU$ of the original matrix A, possibly equilibrated if $EQUED$ is present.

If $FACT$ is present and $FACT = 'F'$, then $IPIV$ is an *input* argument (and must be present); on entry, it must contain the pivot indices from the factorization of A (possibly equilibrated if $EQUED$ is present), returned by a previous call to **LA_GESVX**.

TRANS – *Optional* (*input*) **CHARACTER*1**.

- If $TRANS$ is present, it specifies the form of the system of equations:

  $= 'N'$: $AX = B$ (No transpose)

  $= 'T'$: $A^T X = B$ (Transpose)

  $= 'C'$: $A^H X = B$ (Conjugate transpose)
- otherwise $TRANS = 'N'$ is assumed.

FACT – *Optional* (*input*) **CHARACTER*1**.
Specifies whether or not the factored form of the matrix A is supplied on entry.

- If FACT is present then:

  $= 'N'$: the matrix A will be equilibrated if $EQUED$ is present, then copied to AF and factored.

  $= 'F'$: on entry, AF and $IPIV$ must contain the factored form of A (possibly equilibrated if $EQUED$ is present).
- otherwise $FACT = 'N'$ is assumed.

EQUED – *Optional* (*input/output*) **CHARACTER*1**.

If $FACT$ is not present or $FACT = 'N'$, then $EQUED$ is an *output* argument. If it is present, then the matrix is equilibrated, and on exit $EQUED$ specifies the scaling of A which has actually been performed:

$= 'N'$: No equilibration.

$= 'R'$: Row equilibration, i.e., A has been premultiplied by $diag(R)$; also B has been premultiplied by $diag(R)$ if $TRANS = 'N'$.

$= 'C'$: Column equilibration, i.e., A has been postmultiplied by $diag(C)$; also B has been premultiplied by $diag(C)$ if $TRANS$ = $'T'$ or $'C'$.

$= 'B'$: Both row and column equilibration: combines the effects of $EQUED = 'R'$ and $EQUED = 'C'$.

If $FACT$ is present and $FACT = 'F'$, then $EQUED$ is an *input* argument; if it is present, it specifies the equilibration of A which was performed in a previous call to **LA_GESVX** with $FACT$ not present or $FACT = 'N'$.

R – *Optional* (*input/output*) **REAL** array, shape (:), $size(R) = size(A, 1)$.
R must be present if $EQUED$ is present and $EQUED = 'R'$ or $'B'$; R is not referenced if $EQUED = 'N'$ or $'C'$.

If $FACT$ is not present or $FACT = 'N'$, then R is an *output* argument. If $EQUED = 'R'$ or $'B'$, R returns the row scale factors for equilibrating A.

If $FACT$ is present and $FACT = 'F'$, then R is an *input* argument. If $EQUED = 'R'$ or $'B'$, R must contain the row scale factors for equilibrating A, returned by a previous call to **LA_GESVX**; each element of R must be positive.

C – *Optional* (*input/output*) **REAL** array, shape (:), $size(C) = size(A, 1)$.
C must be present if $EQUED$ is present and $EQUED = 'C'$ or $'B'$; C is not referenced if $EQUED = 'N'$ or $'R'$.

If $FACT$ is not present or $FACT = 'N'$, then C is an *output* argument. If $EQUED = 'C'$ or $'B'$, C returns the column scale factors for equilibrating A.

If $FACT$ is present and $FACT = 'F'$, then C is an *input* argument. If $EQUED = 'C'$ or $'B'$, C must contain the column scale factors for equilibrating A, returned by a previous call to **LA_GESVX**; each element of C must be positive.

FERR – *Optional* (*output*) **REAL** array of shape (:) or **REAL** scalar.
If it is an array, $size(FERR) = size(X, 2)$. The estimated forward error bound for each solution vector $X(j)$ (the j-th column of the solution matrix X). If $XTRUE$ is the true solution corresponding to $X(j)$, $FERR(j)$ is an estimated upper bound for the magnitude of the largest element in $(X(j) - XTRUE)$ divided by the magnitude of the largest element in $X(j)$. The estimate is as reliable as the estimate for $RCOND$, and is almost always a slight overestimate of the true error.

BERR – *Optional* (*output*) **REAL** array of shape (:) or **REAL** scalar.
If it is an array, $size(BERR) = size(X, 2)$. The componentwise relative backward error of each solution vector $X(j)$ (i.e., the smallest relative change in any element of A or B that makes $X(j)$ an exact solution).

RCOND – *Optional* (*output*) **REAL**.
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If $RCOND$ is less than the machine precision (in

particular, if $RCOND = 0$), the matrix is singular to working precision. This condition is indicated by a return code of $INFO > 0$, and the solution and error bounds are not computed.

RPVGRW – *Optional* (*output*) **REAL**.
The reciprocal pivot growth factor $\|A\|_\infty / \|U\|_\infty$. If $RPVGRW$ is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator $RCOND$, and forward error bound $FERR$ could be unreliable. If factorization fails with $0 < INFO \leq size(A,1)$, then $RPVGRW$ contains the reciprocal pivot growth factor for the leading $INFO$ columns of A.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  - $= 0$: successful exit
  - < 0 : if $INFO = -i$, the i-th argument had an illegal value
  - > 0 : if $INFO = i$, and i is
    - $\leq N$: $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed.
    - $= N+1$: $RCOND$ is less than machine precision. The factorization has been completed, but the matrix is singular to working precision, and the solution and error bounds have not been computed.
- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.3 LA_GETRF

A.3.1 Purpose

LA_GETRF computes an LU factorization of a general rectangular matrix A using partial pivoting with row interchanges.

The factorization has the form $A = PLU$ where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$), where $m = size(A,1)$ and $n = size(A,2)$.

When A is square ($m = n$), **LA_GETRF** optionally estimates the reciprocal of the condition number of the matrix A, in either the 1-norm or the ∞-norm. An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $RCOND = 1/(\|A\| \, \|A^{-1}\|)$.

A.3.2 Specification

SUBROUTINE LA_GETRF(A, IPIV, RCOND, NORM, INFO)
  type(*wp*), INTENT(INOUT) :: A(:,:)
  INTEGER, INTENT(OUT) :: IPIV(:)
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: NORM
  REAL(*wp*), INTENT(OUT), OPTIONAL :: RCOND
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  where
  type ::= REAL | COMPLEX
  wp ::= KIND(1.0) | KIND(1.0D0)

A.3.3 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:, :).

- On entry, the matrix A.
- On exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.

IPIV – (*output*) **INTEGER** array, shape (:), $size(IPIV) = \min(size(A,1), size(A,2))$.
Indices that define the permutation matrix P; row i of the matrix was interchanged with row $IPIV(i)$.

RCOND – *Optional* (*output*) **REAL**.
The reciprocal of the condition number of the matrix A for the case $m = n$, computed as $RCOND = 1/(\|A\| \, \|A^{-1}\|)$. $RCOND$ should be present if $NORM$ is present. If $m \neq n$ then $RCOND$ is returned as zero.

NORM – *Optional* (*input*) **CHARACTER*1**.
Specifies whether the 1-norm condition number or the ∞-norm condition number is required:

- = '1', '*O*' or '*o*': 1-norm;
- = '*I*', '*i*': ∞-norm.

If $NORM$ is not present, the 1-norm is used.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  $= 0$: successful exit
  < 0 : if $INFO = -i$, the i-th argument had an illegal value

- > 0 : if $INFO = i$, $U(i,i)]$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.4 LA_GETRS

A.4.1 Purpose

LA_GETRS solves a system of linear equations $AX = B$, $A^T X = B$ or $A^H X = B$ with a general square matrix A, using the LU factorization computed by **LA_GETRF**.

A.4.2 Specification

SUBROUTINE LA_GETRS (A, IPIV, B, TRANS, INFO)
  type(*wp*), INTENT(IN) :: A(:,:)
  INTEGER, INTENT(IN) :: IPIV(:)
  type(*wp*), INTENT(INOUT) :: *rhs*
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: TRANS
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  where
  type ::= REAL | COMPLEX
  wp ::= KIND(1.0) | KIND(1.0D0)
  rhs ::= B(:,:) | B(:)

A.4.3 Arguments

A – (*input*) **REAL** or **COMPLEX** array, shape $(:,:)$, $size(A,1) = size(A,2)$.
The factors L and U from the factorization $A = PLU$ as computed by **LA_GETRF**.

IPIV – (*input*) **INTEGER** array, shape $(:)$, $size(IPIV) = size(A,1)$.
The pivot indices from **LA_GETRF**; for $1 \leq i \leq size(A,1)$, row i of the matrix was interchanged with row $IPIV(i)$.

B – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$ or $(:)$, and $size(B,1)$ or $size(B) = size(A,1)$.

- On entry, the right hand side vector(s) of matrix B for the system of equations $AX = B$.
- On exit, if there is no error, the matrix of solution vector(s) X.

TRANS – *Optional* (*input*) **CHARACTER*1**.

- If $TRANS$ is present, it specifies the form of the system of equations:
  $= 'N'$: $AX = B$ (No transpose)
  $= 'T'$: $A^T X = B$ (Transpose)
  $= 'C'$: $A^H X = B$ (Conjugate transpose)
- otherwise $TRANS = 'N'$ is assumed.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  $= 0$: successful exit
  < 0 : if $INFO = -i$, the i-th argument had an illegal value
- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.5 LA_GETRI

A.5.1 Purpose

LA_GETRI computes the inverse of a matrix using the LU factorization computed by **LA_GETRF**.

A.5.2 Specification

SUBROUTINE LA_GETRI (A, IPIV, INFO)
  type(*wp*), INTENT(INOUT) :: A(:,:)
  INTEGER, INTENT(IN) :: IPIV(:)
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  where
  type ::= REAL | COMPLEX
  wp ::= KIND(1.0) | KIND(1.0D0)

A.5.3 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape $(:,:)$, $size(A,1) = size(A,2)$.

- On entry contains the factors L and U from the factorization $A = PLU$ as computed by **LA_GETRF**.
- On exit, if $INFO = 0$, the inverse of the original matrix A.

IPIV – (*input*) **INTEGER** array, shape (:), $size(IPIV) = size(A,1)$.
The pivot indices from **LA_GETRF**; for $1 \leq i \leq size(A,1)$, row i of the matrix was interchanged with row $IPIV(i)$.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  $= 0$: successful exit
  < 0 : if $INFO = -i$, the i-th argument had an illegal value
  > 0 : if $INFO = i$, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.
- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.6 LA_GERFS

A.6.1 Purpose

LA_GERFS improves the computed solution X of a system of linear equations $AX = B$ or $A^T X = B$ and provides error bounds and backward error estimates for the solution. **LA_GERFS** uses the LU factors computed by **LA_GETRF**.

A.6.2 Specification

SUBROUTINE LA_GERFS (A, AF, IPIV, B, X, &
TRANS, FERR, BERR, INFO)
type(*wp*), INTENT(IN) :: A(:,:), AF(:,:), *rhs*
INTEGER, INTENT(IN) :: IPIV(:)
type(*wp*), INTENT(INOUT) :: *sol*
CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: TRANS
REAL(*wp*), INTENT(OUT), OPTIONAL :: *err*
INTEGER, INTENT(OUT), OPTIONAL :: INFO
where
type ::= REAL | COMPLEX
wp ::= KIND(1.0) | KIND(1.0D0)
rhs ::= B(:,:) | B(:)
sol ::= X(:,:) | X(:)
err ::= FERR(:), BERR(:) | FERR, BERR

A.6.3 Arguments

A – (*input*) **REAL** or **COMPLEX** array, shape (:, :), $size(A,1) = size(A,2)$.
The original matrix A.

AF – (*input*) **REAL** or **COMPLEX** array, shape (:,:), $size(AF,1) = size(AF,2) = size(A,1)$.
The factors L and U from the factorization $A = PLU$ as computed by **LA_GETRF**.

IPIV – (*input*) **INTEGER** array, shape (:), $size(IPIV) = size(A,1)$.
The pivot indices from **LA_GETRF**; for $1 \le i \le size(A,1)$, row i of the matrix was interchanged with row $IPIV(i)$.

B – (*input*) **REAL** or **COMPLEX** array, shape (:,:) or (:), $size(B,1)$ or $size(B) = size(A,1)$.
The right hand side vector(s) of matrix B for the system of equations $AX = B$.

X – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:) or (:), $size(X,1)$ or $size(X) = size(A,1)$.

- On entry, the solution matrix X, as computed by **LA_GETRS**.
- On exit, the improved solution matrix X.

TRANS – *Optional* (*input*) **CHARACTER*1**.

- If $TRANS$ is present, it specifies the form of the system of equations:
  $= 'N'$: $AX = B$ (No transpose)
  $= 'T'$: $A^T X = B$ (Transpose)
  $= 'C'$: $A^H X = B$ (Conjugate transpose)
- otherwise $TRANS = 'N'$ is assumed.

FERR – *Optional* (*output*) **REAL** array of shape (:) or **REAL** scalar.
If it is an array, $size(FERR) = size(X,2)$. The estimated forward error bound for each solution vector $X(j)$ (the j-th column of the solution matrix X). If $XTRUE$ is the true solution corresponding to $X(j)$, $FERR(j)$ is an estimated upper bound for the magnitude of the largest element in $(X(j) - XTRUE)$ divided by the magnitude of the largest element in $X(j)$. The estimate is as reliable as the estimate for $RCOND$, and is almost always a slight overestimate of the true error.

BERR – *Optional* (*output*) **REAL** array of shape (:) or **REAL** scalar.
If it is an array, $size(BERR) = size(X,2)$. The componentwise relative backward error of each solution vector $X(j)$ (i.e., the smallest relative change in any element of A or B that makes $X(j)$ an exact solution).

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present

$= 0$: successful exit

< 0 : if $INFO = -i$, the i-th argument had an illegal value

- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

A.6.4 Internal Parameters

ITMAX – is the maximum number of steps of iterative refinement. It is set to 5 in the **LAPACK 77** subroutines (see [1]).

A.7 LA_GEEQU

A.7.1 Purpose

LA_GEEQU computes row and column scalings intended to equilibrate a rectangle matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest entry in each row and column of the matrix B with elements $B_{ij} = R_i A_{ij} C_j$ have absolute value 1.

R_i and C_j are restricted to be between $SMLNUM$ = smallest safe number and $BIGNUM$ = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

A.7.2 Specification

```
SUBROUTINE LA_GEEQU ( A, R, C, ROWCND, COLCND, &
                AMAX, INFO )
    type(wp), INTENT(IN) :: A(:,:)
    REAL(wp), INTENT(OUT) :: R(:), C(:)
    REAL(wp), INTENT(OUT), OPTIONAL :: ROWCND, &
                COLCND, AMAX
    INTEGER, INTENT(OUT), OPTIONAL :: INFO
    where
    type ::= REAL | COMPLEX
    wp ::= KIND(1.0) | KIND(1.0D0)
```

A.7.3 Arguments

A – (*input*) **REAL** or **COMPLEX** array, shape (:, :).
The matrix A, whose equilibration factors are to be computed.

R – (*output*) **REAL** array, shape (:), $size(R) = size(A, 1)$.
If $INFO = 0$ or $INFO > size(A, 1)$, R contains the row scale factors for A.

C – (*output*) **REAL** array, shape (:), $size(C) = size(A, 2)$.
If $INFO = 0$, C contains the column scale factors for A.

ROWCND – *Optional* (*output*) **REAL**.
If $INFO = 0$ or $INFO > size(A, 1)$, $ROWCND$ contains the ratio of the smallest $R(i)$ to the largest $R(i)$. If $ROWCND \geq 0.1$ and $AMAX$ is neither too large nor too small, it is not worth scaling by R.

COLCND – *Optional* (*output*) **REAL**.
If $INFO = 0$, $COLCND$ contains the ratio of the smallest $C(i)$ to the largest $C(i)$. If $COLCND \geq 0.1$, it is not worth scaling by C.

AMAX – *Optional* (*output*) **REAL**.
Absolute value of largest matrix element. If $AMAX$ is very close to overflow or very close to underflow, the matrix should be scaled.

INFO – *Optional* (*output*) **INTEGER**.

- If $INFO$ is present
  - $= 0$: successful exit
  - < 0 : if $INFO = -i$, the i-th argument had an illegal value
  - > 0 : if $INFO = i$, and i is
    - $\leq m$: the i-th row of A is exactly zero
    - $> m$: the $(i - m)$-th column of A is exactly zero

    where $m = size(A, 1)$.
- If $INFO$ is not present and an error occurs, then the program is terminated with an error message.

B Symmetric and Hermitian Eigenvalue and Eigenvector Procedures, Documentation

B.1 LA_SYEV / LA_HEEV

B.1.1 Purpose

LA_SYEV / LA_HEEV computes all eigenvalues and, optionally, eigenvectors of a real symmetric or complex Hermitian matrix A.

B.1.2 Specification

SUBROUTINE LA_SYEV / LA_HEEV(A, W, JOBZ, UPLO, INFO)
    CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
    INTEGER, INTENT(OUT), OPTIONAL :: INFO
    type(*wp*), INTENT(INOUT) :: A(:,:)
    type(*wp*), INTENT(OUT) :: W(:)
    where
    type ::= REAL | COMPLEX
    wp ::= KIND(1.0) | KIND(1.0D0)

B.1.3 Defaults

- If JOBZ is not present then JOBZ = 'N' is assumed.
- If UPLO is not present then UPLO = 'U' is assumed.

B.1.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:)

- On entry, the symmetric (Hermitian) matrix A.
  - If UPLO = 'U', the upper triangular part of A contains the upper triangular part of the matrix A.
  - If UPLO = 'L', the lower triangular part of A contains
  - the lower triangular part of the matrix A.
- On exit:
  - If JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A.
  - If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

W – (*output*) **REAL** array, shape (:), size(W) = size(A,1) ≥ 0.

- If INFO = 0, the eigenvalues in ascending order.

JOBZ – *Optional*, (*input*) **CHARACTER*1**

- If JOBZ is present then:
  - = 'N': Compute eigenvalues only;
  - = 'V': Compute eigenvalues and eigenvectors.
- otherwise JOBZ = 'N' is assumed.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored;
  - = 'L': Lower triangle of A is stored.
- otherwise UPLO = 'U' is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
  - > 0: if $INFO = i$, the algorithm failed to converge; i form did not converge to zero.
- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.2 LA_SYEVD / LA_HEEVD

B.2.1 Purpose

LA_SYEVD / LA_HEEVD computes all eigenvalues and, optionally, eigenvectors of a real symmetric or complex Hermitian matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

B.2.2 Specification

SUBROUTINE LA_SYEVD / LA_HEEVD(A, W, JOBZ, UPLO, INFO)
CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
INTEGER, INTENT(OUT), OPTIONAL :: INFO
type(*wp*), INTENT(INOUT) :: A(:,:)
type(*wp*), INTENT(OUT) :: W(:)
where
type ::= REAL | COMPLEX
wp ::= KIND(1.0) | KIND(1.0D0)

B.2.3 Defaults

- If JOBZ is not present then JOBZ = 'N' is assumed.
- If UPLO is not present then UPLO = 'U' is assumed.

B.2.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:).

- On entry, the symmetric (Hermitian) matrix A.
  - If UPLO = 'U', the upper triangular part of A contains the upper triangular part of the matrix A.
  - If UPLO = 'L', the lower triangular part of A contains
  - the lower triangular part of the matrix A.
- On exit:
  - If JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A.
  - If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

W – (*output*) **REAL** array, shape (:), size(W) = size(A,1) ≥ 0.

- If INFO = 0, the eigenvalues in ascending order.

JOBZ – *Optional*, (*input*) **CHARACTER*1**

- If JOBZ is present then:
  - = 'N': Compute eigenvalues only;
  - = 'V': Compute eigenvalues and eigenvectors.
- otherwise JOBZ = 'N' is assumed.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored;
  - = 'L': Lower triangle of A is stored.
- otherwise UPLO = 'U' is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
  - > 0: if $INFO = i$, the algorithm failed to converge; i
    form did not converge to zero.
- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.3 LA_SYEVX / LA_HEEVX

B.3.1 Purpose

LA_SYEVX / LA_HEEVX computes selected eigenvalues and, optionally, eigenvectors of a real symmetric or complex Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

B.3.2 Specification

```
SUBROUTINE LA_SYEVX / LA_HEEVX ( A, W, JOBZ, UPLO, VL, VU, &
                IL, IU, M, IFAIL, ABSTOL, INFO )
    CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
    INTEGER, INTENT(IN), OPTIONAL :: IL, IU
    INTEGER, INTENT(OUT), OPTIONAL :: INFO, M
    REAL(wp), INTENT(IN), OPTIONAL :: ABSTOL, VL, VU
    INTEGER, INTENT(OUT), OPTIONAL :: IFAIL(:)
    type(wp), INTENT(INOUT) :: A(:,:)
    REAL(wp), INTENT(OUT) :: W(:)
    where
    type ::= REAL | COMPLEX
    wp ::= KIND(1.0) | KIND(1.0D0)
```

B.3.3 Defaults

- If JOBZ and IFAIL are not present then JOBZ = 'N' is assumed.
- If JOBZ is not present and IFAIL is present then JOBZ = 'V' is assumed.
- If UPLO is not present then UPLO = 'U' is assumed.
- If IL or IU are not present and VL or VU are not present then all eigenvalues are computed (M = size(A,1)).
- If IL is present and IU is not present then IU = size(A,1) is assumed. If IL is not present and IU is present then IL = 1 is assumed.
- If VL is present and VU is not present then VU = +infinity is assumed. If VL is not present and VU is present then VL = -infinity is assumed.
- If ABSTOL is not present then ABSTOL = 2*SLAMCH('S') is assumed. In this case the eigenvalues are computed most accurately.

B.4 Argument dependency

- If either IL or IU are present then neither VL or VU are present.
- If either VL or VU are present then neither IL or IU are present.
- If JOBZ = 'N' then IFAIL must not be present.

B.4.1 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:), size(A,1) = size(A,2) ≥ 0.

- On entry, the symmetric (Hermitian) matrix A.
  - If UPLO = 'U', the upper triangular part of A contains the upper triangular part of the matrix A.
  - If UPLO = 'L', the lower triangular part of A contains the lower triangular part of the matrix A.
- On exit:
  - If JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of A holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of A contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.

- If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

W – (*output*) **REAL** array, shape (:), size(W) = size(A,1) $\geq$ 0.

- On normal exit, the first M elements contain the selected eigenvalues in ascending order.

JOBZ – *Optional*, (*input*) **CHARACTER*1**

- If JOBZ is present then:
  - $=' N'$: Compute eigenvalues only;
  - $=' V'$: Compute eigenvalues and eigenvectors.
- otherwise JOBZ = 'N' is assumed.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - $=' U'$: Upper triangle of A is stored;
  - $=' L'$: Lower triangle of A is stored.
- otherwise $UPLO =' U'$ is assumed.

VL – *Optional*, (*input*) **REAL**.

VU – *Optional*, (*input*) **REAL**.

- If VL and VU are present ($VL < VU$) then the lower and upper bounds of the interval to be searched for eigenvalues. All eigenvalues in the half-open interval (VL,VU] will be found.

IL – *Optional*, (*input*) **INTEGER**.

IU – *Optional*, (*input*) **INTEGER**.

- If IL and IU are present ($1 \leq IL \leq IU \leq size(A,1)$) then the indices (in ascending order) of the smallest and largest eigenvalues to be returned. The IL^{th} through IU^{th} eigenvalues will be found.

M – *Optional*, (*output*) **INTEGER**.

- The total number of eigenvalues found ($0 \leq M \leq size(A,1)$).
- If IL and IU are present then $M = IU - IL + 1$.

IFAIL – *Optional*, (*output*) **INTEGER** array, shape (:), $size(IFAIL) = size(A,1) \geq 0$.

- If IFAIL is present then $JOBZ =' V'$ is assumed and eigenvectors are computed.
  - If $INFO = 0$, the first M elements of IFAIL are zero.
  - If $INFO > 0$, then IFAIL contains the indices of the eigenvectors that failed to converge.

ABSTOL – *Optional*, (*input*) **REAL**.

- The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

  $$ABSTOL + EPS \times \max(|a|, |b|)$$

  ,
  where EPS is the machine precision. If $ABSTOL$ is less than or equal to zero, then $EPS \times |T|r$ will be used in its place, where $|T|$ is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.
- Eigenvalues will be computed most accurately when $ABSTOL$ is set to twice the underflow threshold $2 \times SLAMCH('S')$, not zero. If this routine returns with $INFO > 0$, indicating that some eigenvectors did not converge, try setting $ABSTOL$ to $2 \times SLAMCH('S')$.
- See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.
- If ABSTOL is not present then $ABSTOL = 2 \times SLAMCH('S')$ is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - $= 0$: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
  - > 0: if $INFO = i$, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.
- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.5 LA_SYGST / LA_HEGST

B.5.1 Purpose

LA_SYGST / LA_HEGST reduces a real symmetric-definite or complex Hermitian-definite generalized eigenproblem to standard form.

- If $ITYPE = 1$, the problem is $Ax = \lambda Bx$, and A is overwritten by $(U^H)^{-1}AU^{-1}$ or $L^{-1}A(L^H)^{-1}$
- If $ITYPE = 2$ or 3, the problem is $ABx = \lambda x$ or $BAx = \lambda x$, and A is overwritten by UAU^H or $L^H AL$.
- B must have been previously factorized as $U^H U$ or LL^H by LA_POTRF.

B.5.2 Specification

SUBROUTINE LA_SYGST / LA_HEGST(A, B, ITYPE, UPLO, INFO)
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: UPLO
  INTEGER, INTENT(IN), OPTIONAL :: ITYPE
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  type(*wp*), INTENT(IN) :: B(:,:)
  type(*wp*), INTENT(INOUT) :: A(:,:)
  where
  type ::= REAL | COMPLEX
  wp ::= KIND(1.0) | KIND(1.0D0)

B.5.3 Defaults

- If ITYPE is not present then $ITYPE = 1$ is assumed.
- If UPLO is not present then UPLO = 'U' is assumed.

B.5.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:),
  $size(A, 1) = size(A, 2) \geq 0$.

- On entry, the symmetric (Hermitian) matrix A.
  - If $UPLO =' U'$, the upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.
  - If $UPLO =' L'$, the lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

- On exit, if $INFO = 0$, the transformed matrix, stored in the same format as A.

B – (*input*) **REAL** or **COMPLEX** array, shape (:,:), $size(B,1) = size(A,1)$.

- The triangular factor from the Cholesky factorization of B, as returned by LA_POTRF.

ITYPE – *Optional*, (*input*) **INTEGER**

- If ITYPE is present then:
  - $= 1$: compute $(U^H)^{-1}AU^{-1}$ or $L^{-1}A(L^H)^{-1}$;
  - $= 2$ or 3: compute UAU^H or L^HAL.
- otherwise $ITYPE = 1$ is assumed.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - $=' U'$: Upper triangle of A is stored and B is factored as U^HU;
  - $=' L'$: Lower triangle of A is stored and B is factored as LL^H.
- otherwise $UPLO =' U'$ is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - $= 0$: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.6 LA_SYGV / LA_HEGV

B.6.1 Purpose

LA_SYGV / LA_HEGV computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite or complex Hermitian-definite eigenproblem, of the form $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$. Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite.

B.6.2 Specification

SUBROUTINE LA_SYGV / LA_HEGV(A, B, W, ITYPE, JOBZ, UPLO, &
INFO)
CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
INTEGER, INTENT(IN), OPTIONAL :: ITYPE
INTEGER, INTENT(OUT), OPTIONAL :: INFO
type(*wp*), INTENT(INOUT) :: A(:,:), B(:,:)
REAL(*wp*), INTENT(OUT) :: W(:)
where
type ::= REAL | COMPLEX
wp ::= KIND(1.0) | KIND(1.0D0)

B.6.3 Defaults

- If ITYPE is not present then $ITYPE = 1$ is assumed.
- If JOBZ is not present then $JOBZ =' N'$ is assumed.
- If UPLO is not present then $UPLO =' U'$ is assumed.

B.6.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:), $size(A,1) = size(A,2) \geq 0$.

- On entry, the symmetric (Hermitian) matrix A.
  - If $UPLO =' U'$, the upper triangular part of A contains the upper triangular part of the matrix A.
  - If $UPLO =' L'$, the lower triangular part of A contains the lower triangular part of the matrix A.
- On exit:
  - If $JOBZ =' V'$, then if $INFO = 0$, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:
    * if $ITYPE = 1$ or 2, $Z^H BZ = I$;
    * if $ITYPE = 3$, $Z^H B^{-1} Z = I$.
  - If $JOBZ =' N'$, then on exit the upper triangle (if $UPLO =' U'$) or the lower triangle (if $UPLO =' L'$) of A, including the diagonal, is destroyed.

B – (*input*) **REAL** or **COMPLEX** array, shape (:,:), $size(B,1) = size(A,1)$.

- On entry, the symmetric (Hermitian) matrix B.

- If $UPLO =' U'$, the upper triangular part of B contains the upper triangular part of the matrix B.
- If $UPLO =' L'$, the lower triangular part of B contains the lower triangular part of the matrix B.

- On exit, if $INFO \leq size(A,1)$, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization $B = U^H U$ or $B = LL^H$.

W – (*output*) **REAL** array, shape (:), $size(W) = size(A,1)$.

- If $INFO = 0$, the eigenvalues in ascending order.

ITYPE – *Optional*, (*input*) **INTEGER**.
Specifies the problem type to be solved.

- If ITYPE is present then:
  - = 1: $Ax = \lambda Bx$
  - = 2: $ABx = \lambda x$
  - = 3: $BAx = \lambda x$
- otherwise $ITYPE = 1$ is assumed.

JOBZ – *Optional*, (*input*) **CHARACTER*1**

- If JOBZ is present then:
  - = 'N': Compute eigenvalues only;
  - = 'V': Compute eigenvalues and eigenvectors.
- otherwise $JOBZ =' N'$ is assumed.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored and B is factored as $U^H U$;
  - = 'L': Lower triangle of A is stored and B is factored as LL^H.
- otherwise $UPLO =' U'$ is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
  - > 0: LA_POTRF or LA_SYEV / LA_HEEV returned an error code:

* $\leq size(A,1)$: if $INFO = i$, LA_SYEV / LA_HEEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
* $> size(A,1)$: if $INFO = size(A,1) + i \leq 2size(A,1)$, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.7 LA_SYTRD / LA_HETRD

B.7.1 Purpose

LA_SYTRD / LA_HETRD reduces a real symmetric or complex Hermitian matrix A to real symmetric tridiagonal form T by an orthogonal or unitary similarity transformation: $Q^H AQ = T$.

B.7.2 Specification

SUBROUTINE LA_SYTRD / LA_HETRD(A, TAU, UPLO, INFO)
    CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: UPLO
    INTEGER, INTENT(OUT), OPTIONAL :: INFO
    type(*wp*), INTENT(INOUT) :: A(:,:)
    type(*wp*), INTENT(OUT) :: TAU(:)
    where
    type ::= REAL | COMPLEX
    wp ::= KIND(1.0) | KIND(1.0D0)

B.7.3 Defaults

- If UPLO is not present then UPLO = 'U' is assumed.

B.7.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:),
$size(A,1) = size(A,2) \geq 0$.

- On entry, the symmetric (Hermitian) matrix A.
  - If UPLO = 'U', the upper triangular part of A contains the upper triangular part of the matrix A.
  - If UPLO = 'L', the lower triangular part of A contains the lower triangular part of the matrix A.

- On exit:
  - If UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors.
  - If UPLO = 'L', the diagonal and first subdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors.
- See Further Details.

TAU – (*output*) **REAL** or **COMPLEX** array, shape (:), $size(TAU) = size(A, 1) - 1$.

- The scalar factors of the elementary reflectors.
- See Further Details.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored
  - = 'L': Lower triangle of A is stored
- otherwise UPLO = 'U' is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
- If INFO is not present and an error occurs, then the program is terminated with an error message.

B.7.5 Further Details

- If UPLO = 'U', the matrix Q is represented as a product of elementary reflectors $Q = H_{n-1} \cdots H_2 H_1$. Each H_i has the form $H_i = I - \tau v v'$, where τ is a complex scalar, and v is a complex vector with $v_{i+1:n} = 0$ and $v_i = 1$; $v_{1:i-1}$ is stored on exit in $A(1 : i-1, i+1)$, and τ in $TAU(i)$.
- If UPLO = 'L', the matrix Q is represented as a product of elementary reflectors $Q = H_1 H_2 \cdots H_{n-1}$. Each H_i has the form $H_i = I - \tau v v'$, where τ is a complex scalar, and v is a complex vector with $v_{1:i} = 0$ and $v_{i+1} = 1$; $v_{i+2:n}$ is stored on exit in $A(i+2 : n, i)$, and τ in $TAU(i)$.

The contents of A on exit are illustrated by the following examples with $n = 5$:

```
    if UPLO = 'U':                        if UPLO = 'L':

( d   e   v2  v3  v4 )              ( d                  )
(     d   e   v3  v4 )              ( e   d              )
(         d   e   v4 )              ( v1  e   d          )
(             d   e  )              ( v1  v2  e   d      )
(                 d  )              ( v1  v2  v3  e   d  )
```

where d and e denote diagonal and off-diagonal elements of T, and v_i denotes an element of the vector defining H_i.

B.8 LA_ORGTR / LA_UNGTR

B.8.1 Purpose

LA_ORGTR / LA_UNGTR generates a real orthogonal / complex unitary matrix Q which is defined as the product of elementary reflectors, as returned by LA_SYTRD / LA_HETRD:

- if UPLO = 'U', $Q = H_{n-1} \cdots H_2 H_1$,
- if UPLO = 'L', $Q = H_1 H_2 \cdots H_{n-1}$.

B.8.2 Specification

SUBROUTINE LA_ORGTR / LA_UNGTR(A, TAU, UPLO, INFO)
    CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: UPLO
    INTEGER, INTENT(OUT), OPTIONAL :: INFO
    type(*wp*), INTENT(IN) :: TAU(:)
    type(*wp*), INTENT(INOUT) :: A(:,:)
    where
    type ::= REAL | COMPLEX
    wp ::= KIND(1.0) | KIND(1.0D0)

B.8.3 Defaults

- If UPLO is not present then UPLO = 'U' is assumed.

B.8.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:), $size(A,1) = size(A,2) \geq 0$.

- On entry, the vectors which define the elementary reflectors, as returned by LA_SYTRD or LA_HETRD.
- On exit the orthogonal or unitary matrix Q.

TAU – (*input*) **REAL** or **COMPLEX** array, shape (:), $size(TAU) = size(A,1) - 1$.

- $TAU(i)$ must contain the scalar factor of the elementary reflector H_i, as returned by LA_SYTRD or LA_HETRD.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored
  - = 'L': Lower triangle of A is stored
- otherwise UPLO = 'U' is assumed.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
- If INFO is not present and an error occurs, then the program is terminated with an error message.

C Cholesky Factorization of a Real Symmetric or Complex Hermitian Positive Definite Matrix A, Documentation

C.1 LA_POTRF

C.1.1 Purpose

LA_POTRF computes the Cholesky factorization of a real symmetric or complex Hermitian positive definite matrix A.

The factorization has the form

- $A = U^H U$, if UPLO = 'U', or
- $A = LL^H$, if UPLO = 'L',

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

This is the block version of the algorithm, calling Level 3 BLAS.

LA_POTRF optionally estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric or complex Hermitian positive definite matrix A. An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $RCOND = 1/\|A\|\|A^{-1}\|$.

C.1.2 Specification

SUBROUTINE LA_POTRF(A, UPLO, RCOND, NORM, INFO)
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: NORM, UPLO
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  REAL(*wp*), INTENT(OUT), OPTIONAL :: RCOND
  type(*wp*), INTENT(INOUT) :: A(:,:)
  where
  type ::= REAL | COMPLEX
  wp ::= KIND(1.0) | KIND(1.0D0)

C.1.3 Defaults

- If UPLO is not present then UPLO = 'U' is assumed.

C.1.4 Arguments

A – (*input/output*) **REAL** or **COMPLEX** array, shape (:,:), $size(A,1) = size(A,2) \geq 0$.

  - On entry, the symmetric (Hermitian) matrix A.

- If UPLO = 'U', the upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.
- If UPLO = 'L', the lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

- On exit, if $INFO = 0$, the factor U or L from the Cholesky factorization $A = U^H U$ or $A = LL^H$.

UPLO – *Optional*, (*input*) **CHARACTER*1**

- If UPLO is present then:
  - = 'U': Upper triangle of A is stored;
  - = 'L': Lower triangle of A is stored.
- otherwise UPLO = 'U' is assumed.

RCOND – *Optional* (*output*) **REAL**

- The reciprocal of the condition number of the matrix A computed as $RCOND = 1/\|A\|\|A^{-1}\|$.

NORM – *Optional* (*input*) **CHARACTER*1**
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:

- If NORM is present then:
  - = '1', 'O' or 'o': 1-norm;
  - = 'I' or 'i': infinity-norm.
- otherwise NORM = '1' is used.

INFO – *Optional*, (*output*) **INTEGER**

- If INFO is present:
  - = 0: successful exit
  - < 0: if $INFO = -i$, the i-th argument had an illegal value
  - > 0: if $INFO = i$, the leading minor of order i is not positive definite, and the factorization could not be completed.
- If INFO is not present and an error occurs, then the program is terminated with an error message.

D Code for One Version of LA_SYEV

We illustrate here the sort of code that is needed to implement one of the Fortran 90 jacket procedures. The procedure shown is the real single precision version of **LA_SYEV**.

D.1 Precision-dependencies

To handle different precisions, we use a module `LA_PRECISION` to define named constants `SP` and `DP` for the kind values of single and double precision, respectively.

```
MODULE LA_PRECISION
  INTEGER, PARAMETER :: SP=KIND(1.0), DP=KIND(1.0D0)
END MODULE LA_PRECISION
```

Within the LAPACK 90 code, all real and complex constructs are expressed in terms of a symbolic kind value `WP`, which is defined by reference to the module `LA_PRECISION` — in single precision:

```
USE LA_PRECISION :: WP => SP
```

and in double precision:

```
USE LA_PRECISION :: WP => DP
```

These are the only precision-dependent changes in the code, apart from changes to the procedure-names.

D.2 Error-handling

To handle errors, as described in Section 4, we use a simple procedure `ERINFO`, which is assumed to be accessed from a module `LA_AUXMOD`:

```
SUBROUTINE ERINFO(LINFO, SRNAME, INFO, ISTAT)
!  .. Scalar Arguments ..
   CHARACTER( LEN = * ), INTENT(IN)              :: SRNAME
   INTEGER             , INTENT(IN)              :: LINFO
   INTEGER             , INTENT(OUT), OPTIONAL   :: INFO
   INTEGER             , INTENT(IN), OPTIONAL    :: ISTAT
!
!  .. Executable Statements ..
!
   IF( ( LINFO < 0 .AND. LINFO > -200 ) .OR. &
       ( LINFO > 0 .AND. .NOT.PRESENT(INFO) ) )THEN
      WRITE (*,*) 'Program terminated in LAPACK_90 subroutine ',SRNAME
```

```
      WRITE (*,*) 'Error indicator, INFO = ',LINFO
      IF( PRESENT(ISTAT) )THEN
         IF( ISTAT /= 0 ) THEN
            IF( LINFO == -100 )THEN
               WRITE (*,*) 'The statement ALLOCATE causes STATUS = ', ISTAT
            ELSE
               WRITE (*,*) 'LINFO = ', LINFO, ' not expected'
            END IF
         END IF
      END IF
      STOP
   ELSE IF( LINFO <= -200 ) THEN
      WRITE(*,*) '++++++++++++++++++++++++++++++++++++++++++++++++'
      WRITE(*,*) '*** WARNING, INFO = ', LINFO, ' WARNING ***'
      IF( LINFO == -200 THEN
         WRITE(*,*) 'Could not allocate sufficient workspace for the optimum'
         WRITE(*,*) 'blocksize, hence the routine may not have performed as'
         WRITE(*,*) 'efficiently as possible'
      ELSE
         WRITE(*.*) 'Unexpected warning'
      END IF
      WRITE(*,*) '++++++++++++++++++++++++++++++++++++++++++++++++'
   END IF
   IF( PRESENT(INFO) ) INFO = LINFO
END SUBROUTINE ERINFO
```

A more elaborate error-handling mechanism could of course be devised.

D.3 Accessing LAPACK 77 routines

We assume that interface-blocks for all the LAPACK 77 routines are accessible from modules `LA_SF77MOD`, `LA_DF77MOD`, `LA_CF77MOD`, and `LA_ZF77MOD`. Note that we do not use generic interfaces for the LAPACK 77 routines, since that would impose some restrictions on the way in which LAPACK 77 routines could be called.

However, we rename the routine in the `USE` statement, so that the precision-dependent name-change is localized in the `USE` statement.

D.4 The code

```
MODULE LA_SSYEV
CONTAINS
!
SUBROUTINE SSYEV_F90( A, W, JOBZ, UPLO, INFO )
```

```
!  .. Use Statements ..
   USE LA_PRECISION, ONLY: WP => SP
   USE LA_AUXMOD, ONLY: ERINFO, LSAME
   USE LA_AUF77MOD, ONLY: ILAENV_F77 => ILAENV
   USE LA_SF77MOD, ONLY: SYEV_F77 => SSYEV
!  .. Implicit Statement ..
   IMPLICIT NONE
!  .. Character Arguments ..
   CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
!  .. Scalar Arguments ..
   INTEGER, INTENT(OUT), OPTIONAL :: INFO
!  .. Array Arguments ..
   REAL(WP), INTENT(INOUT) :: A(:,:)
   REAL(WP), INTENT(OUT) :: W(:)
!  .. Local Parameters ..
   CHARACTER(LEN=7), PARAMETER :: SRNAME = 'LA_SYEV'
   CHARACTER(LEN=6), PARAMETER :: BSNAME = 'SSYTRD'
!  .. Local Scalars ..
   CHARACTER(LEN=1) :: LJOBZ, LUPLO
   INTEGER :: N, LINFO, LD, ISTAT, ISTAT1, LWORK, NB
!  .. Local Arrays ..
   REAL(WP), POINTER :: WORK(:)
!  .. Intrinsic Functions ..
   INTRINSIC MAX, PRESENT
!  .. Executable Statements ..
   N = SIZE( A, 1 ); LINFO = 0; ISTAT = 0; LD = MAX(1,N)
   IF( PRESENT(JOBZ) ) THEN
      LJOBZ = JOBZ
   ELSE
      LJOBZ = 'N'
   END IF
   IF( PRESENT(UPLO) ) THEN
      LUPLO = UPLO
   ELSE
      LUPLO = 'U'
   END IF
!  .. Test the arguments
   IF( SIZE( A, 2 ) /= N .OR. N < 0 )THEN
      LINFO = -1
   ELSE IF( SIZE( W ) /= N )THEN
      LINFO = -2
   ELSE IF( .NOT.LSAME(LJOBZ,'N') .AND. .NOT.LSAME(LJOBZ,'V') )THEN
      LINFO = -3
   ELSE IF( .NOT.LSAME(LUPLO,'U') .AND. .NOT.LSAME(LUPLO,'L') )THEN
```

```
      LINFO = -4
   ELSE IF( N > 0 )THEN
!  .. Determine the workspace
      NB = ILAENV_F77( 1, BSNAME, LUPLO, N, -1, -1, -1 )
      IF( NB <= 1 .OR. NB >= N )THEN
         NB = 1
      END IF
      LWORK = (2+NB)*N
      ALLOCATE(WORK(LWORK), STAT=ISTAT)
      IF( ISTAT /= 0 )THEN
         LWORK = 3*N-1
         ALLOCATE(WORK(LWORK), STAT=ISTAT)
         IF( ISTAT /= 0 ) THEN
            LINFO = - 100
         ELSE
            LINFO = - 200
         ENDIF
      ENDIF
!
      IF( LINFO == 0 .OR. LINFO <= -200 )THEN
!        .. Call LAPACK77 routine
         CALL SYEV_F77( LJOBZ, LUPLO, N, A, LD, W, WORK, LWORK, LINFO )
      ENDIF
      DEALLOCATE(WORK, STAT=ISTAT1)
   ENDIF
   CALL ERINFO(LINFO,SRNAME,INFO,ISTAT)
END SUBROUTINE SSYEV_F90
!
END MODULE LA_SSYEV
```

D.5 Accessing LAPACK 90 procedures

We assume that interface-blocks (module-procedures) for all the LAPACK 90 routines are accessible from module LA_SCDZF90MOD.