LAPACK Working Note 101
A Proposal for a Fortran 90 Interface for LAPACK

Jack J. Dongarra* Jeremy Du Croz† Sven Hammarling†
Jerzy Waśniewski‡ Adam Zemla§

August 10, 1995

1 Introduction

The purpose of this paper is to initiate discussion of the design of a Fortran 90 interface to LAPACK [1]. 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 new interface can be implemented initially by writing Fortran 90 jackets to call the existing Fortran 77 code.

Eventually we hope that the LAPACK code will be rewritten to take advantage of the new features of Fortran 90, but this will be an enormous task. We aim to design an interface which can persist unchanged while the underlying code is rewritten.

For convenience we use the name “LAPACK77” to denote the existing Fortran 77 package, and “LAPACK90” to denote the new Fortran 90 interface which we are proposing.

2 LAPACK77 and Fortran 90 Compilers

2.1 Linking LAPACK77 to Fortran 90 programs

LAPACK77 can be called from Fortran 90 programs in its present form — with some qualifications. The qualifications arise only because LAPACK77 is not written entirely in

*Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301 and Mathematical Sciences Section, Oak Ridge National Laboratory, P.O.Box 2008, Bldg. 6012; Oak Ridge, TN 37831-6367, Email: dongarra@cs.utk.edu
†Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK, Email: jeremy@nag.co.uk or sven@nag.co.uk respectively
‡UNiC, Bldg. 304, Technical University of Denmark, DK-2800 Lyngby, Denmark, Email: jerzy.wasniewski@unic.dk
§Institute of Mathematics, Polish Academy of Sciences, Sniadeckich 8, 00-950 Warsaw, Poland, Email: adamz@impan.gov.pl
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 LAPACK77 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 LAPACK77 library, compiled by a Fortran 77 compiler. This approach can only work if the compilers have designed to allow cross-linking.

2. If such cross-linking is not possible, recompile LAPACK77 with the Fortran 90 compiler, provided that the compiler accepts COMPLEX*16 and related intrinsics as extensions, and create a new library.

3. If these extensions are not accepted, convert the LAPACK77 code to standard Fortran 90 (see below), before recompiling it.

The conversions needed to create standard Fortran 90 code for LAPACK77 are:

\[
\begin{align*}
\text{COMPLEX*16} & \quad \Rightarrow \quad \text{COMPLEX(KIND=Kind(0.0D0)} \\
\text{DCONJG}(z) \text{ for COMPLEX*16 } z & \quad \Rightarrow \quad \text{CONJG}(z) \\
\text{DBLE}(z) \text{ for COMPLEX*16 } z & \quad \Rightarrow \quad \text{REAL}(z) \\
\text{DIMAG}(z) \text{ for COMPLEX*16 } z & \quad \Rightarrow \quad \text{AIMAG}(z) \\
\text{DCMPLX}(x,y) \text{ for DOUBLE PRECISION } x, y & \quad \Rightarrow \quad \text{CMPLX}(x,y,KIND=\text{Kind(0.0D0)})
\end{align*}
\]

One further obstacle may remain: it is possible that if LAPACK77 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 LAPACK77

Fortran 90 allows one immediate extra benefit to be provided to Fortran 90 users of LAPACK77, 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, as illustrated by the following example (in fixed-form source format) containing an interface for the single routine CBDSQR:

\[
\begin{align*}
\text{MODULE LAPACK77_INTERFACES}
\end{align*}
\]
A single module containing interfaces for all the routines in LAPACK77 (over 1000 of them) may be too large for practical use; it may be desirable to split it (perhaps, one module for single precision documented routines, one for double precision documented routines, and similarly for auxiliary routines).

3 Proposals for the Design of LAPACK90

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 LAPACK90 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:

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

   - passing array sections, for example:

     ```fortran
     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 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. This point needs investigation.
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 LAPACK77, 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 LAPACK77 routine DGEEV, are not needed in LAPACK90.

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 LAPACK77. For example, in LAPACK77, 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 LAPACK90, 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:

(a) the initial letter (S, C, D or Z) is simply omitted.

(b) 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 through the
use of array pointers (which such derived types would require).

6. Error-handling:
In LAPACK77, 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:** the routine sets INFO<0 and calls the
auxiliary routine XERBLA; the standard version of XERBLA issues an error mes-
 sage 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 LAPACK77
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 LAPACK90 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 LAPACK90 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 LAPACK77. Note however that in Fortran 90 there can be different
reasons for an argument being invalid:

**illegal value:** as in LAPACK77.

**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.

The specification could be extended so that the error-message could distinguish be-
tween these cases.

4 Prototype Implementation of LAPACK90 Procedures

We have implemented Fortran 90 jacket procedures to the group of LAPACK77 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.
In Appendix A, we give detailed documentation of the proposed interfaces. Here we give examples of calls to each of the proposed routines, the first without using any of the optional arguments, the second using all the arguments. For the time being and for ease of comparison between LAPACK77 and LAPACK90, 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 LAPACK77 is not modified.

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 LAPACK77).
- 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):

```fortran
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):

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

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

```fortran
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):

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

## 5 Documentation

In Appendix A, 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 will be needed to test the new interfaces.
7 Timings

We have done some timings to measure the extra overhead of the Fortran 90 interface. We timed \texttt{LA\_GETRF} on a single processor of an IBM SP-2 (in double precision) and a single processor of a Cray YMP C92 (in single precision). All timings are given in megaflops.

**IBM**
1. Speed of LAPACK90 calling LAPACK77 and BLAS from the ESSL library.
2. Speed of LAPACK77, using BLAS from the ESSL library.

<table>
<thead>
<tr>
<th>Array size</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK90</td>
<td>187</td>
<td>180</td>
<td>182</td>
<td>170</td>
<td>172</td>
<td>172</td>
<td>176</td>
<td>177</td>
<td>181</td>
<td>182</td>
</tr>
<tr>
<td>LAPACK77</td>
<td>191</td>
<td>181</td>
<td>182</td>
<td>171</td>
<td>172</td>
<td>173</td>
<td>176</td>
<td>179</td>
<td>180</td>
<td>182</td>
</tr>
</tbody>
</table>

**Cray**
1. Speed of LAPACK90 calling LAPACK77 as provided by CRAY in LIBSCI.
2. Speed of LAPACK77 as provided by CRAY in LIBSCI.

<table>
<thead>
<tr>
<th>Array size</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK90</td>
<td>260</td>
<td>781</td>
<td>260</td>
<td>812</td>
<td>266</td>
<td>832</td>
<td>268</td>
<td>847</td>
<td>270</td>
<td>856</td>
</tr>
<tr>
<td>LAPACK77</td>
<td>261</td>
<td>787</td>
<td>261</td>
<td>817</td>
<td>267</td>
<td>838</td>
<td>268</td>
<td>851</td>
<td>271</td>
<td>859</td>
</tr>
</tbody>
</table>

Two conclusions can be taken from the above tables:

1. The LAPACK90 results can be a little slower (1 or 2\%) than the LAPACK77 results.

2. The Cray timings show a striking loss of speed for matrix sizes which are a multiple of 8. In these tests, the leading dimension of the array holding the matrix was equal to the matrix size (as naturally happens when using assumed-shape arrays); when the leading dimension is a multiple of 8, memory bank conflicts slow the computations down. A more careful implementation of the BLAS might be able to reduce this sensitivity to the value of the leading dimension. In Fortran 77, a user can always avoid the problem by declaring arrays with odd values for the leading dimension. In Fortran 90, the situation is more complicated; with the current implementation of LAPACK90, it is likely that the compiler will pass a temporary copy of the array \( A \) (with leading dimension \( N \)) to the LAPACK77 code where all the intensive work is done – so users would have no means of avoiding memory bank conflict in the LAPACK77 code. If Fortran 90 code were used throughout then assumed-shape arrays could be used everywhere and the compiler would not need to create any temporary copies, and users would have more control.

8 Acknowledgments

Jerzy Waśniewski’s research is partly supported by the Danish Project, Efficient Parallel Algorithms for Optimization and Simulation (EPOS).

We thank very much Dr. Christian de Polignac. He ran the test programs on an IBM RS/6000 and using the NAG compiler on an HP workstation.
References


A Documentation of LAPACK90 Procedures

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

A.1.3 Arguments

A — (input/output) either REAL or COMPLEX square 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) either REAL or COMPLEX rectangular array, shape either (,:,:) or (:,:), $size(B, 1)$ 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
\geq 0 : if INFO = k, U(k, k) 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 either 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' : \( \text{diag}(R) A \text{diag}(C) \text{diag}(C)^{-1} X = \text{diag}(R) B \)
   TRANS = 'T' : \( (\text{diag}(R) A \text{diag}(C))^T \text{diag}(R)^{-1} X = \text{diag}(C) B \)
   TRANS = 'C' : \( (\text{diag}(R) A \text{diag}(C))^H \text{diag}(R)^{-1} X = \text{diag}(C) B \)

   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 \( \text{diag}(R) A \text{diag}(C) \) and B by \( \text{diag}(R) B \) (if TRANS = 'N') or \( \text{diag}(C) B \) (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 $\text{diag}(C)$ (if $\text{TRANS} = 'N'$) or $\text{diag}(R)$ (if $\text{TRANS} = 'T'$ or 'C') so that it solves the original system before equilibration.

A.2.4 Arguments

**A** — *(input/output)* either REAL or COMPLEX square array, shape $(\cdot, \cdot)$, $\text{size}(A,1) = \text{size}(A,2)$.

If $\text{FACT}$ is not present or $\text{FACT} = 'N'$,
- On entry, the matrix $A$.
- On exit, if $\text{EQUED}$ is present, the matrix $A$ may have been overwritten by the equilibrated matrix (see $\text{EQUED}$).

If $\text{FACT}$ is present and $\text{FACT} = 'F'$,
- On entry, the matrix $A$, possibly equilibrated in a previous call to LA_GESVX (see $\text{EQUED}$).
- On exit, $A$ is unchanged.

**B** — *(input/output)* either REAL or COMPLEX rectangular array, shape either $(\cdot, \cdot)$ or $(\cdot)$, $\text{size}(B,1) = \text{size}(A,1)$.

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

**X** — *(output)* either REAL or COMPLEX rectangular array, shape either $(\cdot, \cdot)$ or $(\cdot)$, $\text{size}(X,1) = \text{size}(A,1)$. If $\text{INFO} = 0$, 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 ($\text{EQUED}$ is present and $\text{EQUED} \neq 'N'$), this does not correspond to the scaled $A$ and $B$.

**AF** — *(Optional input/output)* either REAL or COMPLEX square array, shape $(\cdot, \cdot)$, $\text{size}(AF,1) = \text{size}(AF,2) = \text{size}(A,1)$.

If $\text{FACT}$ is not present or $\text{FACT} = 'N'$, 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 $\text{EQUED}$ is present.

If $\text{FACT}$ is present and $\text{FACT} = 'F'$, 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 $\text{EQUED}$ is present), returned by a previous call to LA_GESVX.

**IPIV** — *(Optional input/output)* INTEGER array, shape $(\cdot)$, $\text{size}(IPIV) = \text{size}(A,1)$.

If $\text{FACT}$ is not present or $\text{FACT} = 'N'$, then $IPIV$ is an output argument and returns the pivot indices from the factorization $A = PLU$ of the original matrix $A$, possibly equilibrated if $\text{EQUED}$ 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'; R 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 \text{LA.GESVX}; each element of C must be positive.

FERR — Optional (output) either 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) either 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 ≤ 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 $m = \text{size}(A, 1)$ and $n = \text{size}(A, 2)$.

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 $\infty$-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) either 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 (:), $\text{size}(IPIV) = \min(\text{size}(A, 1), \text{size}(A, 2))$. 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 $\infty$-norm condition number is required:
   - = '1', 'O' or 'o': 1-norm;
   - = 'I', 'i': $\infty$-norm.
If $NORM$ is not present, the 1-norm is used.

INFO — Optional (output) INTEGER.

- If $INFO$ is present
  - $= 0$ : successful exit
  - $< 0$ : if $INFO = -k$, the $k$-th argument had an illegal value
  - $> 0$ :
    - if $INFO = k$, $U(k,k)$ 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 $AX = B$, $A^TX = B$ or $AHX = B$ 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) either REAL or COMPLEX square 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) either REAL or COMPLEX rectangular array, shape either (:,:) or (:), $size(B,1)$ 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 $INFO = -k$, the $k$-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

```fortran
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) either REAL or COMPLEX square 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 $A$.  

---

18
**IPIV** – *(input)* **INTEGER** array, shape (:), \(\text{size}(\text{IPIV}) = \text{size}(A, 1)\). The pivot indices from **LA_GETRF**; for \(1 \leq i \leq \text{size}(A, 1)\), row \(i\) of the matrix was interchanged with row \(\text{IPIV}(i)\).

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

- If **INFO** is present
  
  = 0 : successful exit
  
  < 0 : if \(\text{INFO} = -k\), the \(k\)-th argument had an illegal value
  
  > 0 : if \(\text{INFO} = k\), \(U(K,K)\) 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 \(A^TX = B\) 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)

\[\text{type}(\text{wp}), \text{INTENT}(\text{IN}) :: A(\text{\ldots}), AF(\text{\ldots}), \text{rhs}\]

\[\text{INTEGER}, \text{INTENT}(\text{IN}) :: \text{IPIV}(\text{\ldots})\]

\[\text{type}(\text{wp}), \text{INTENT}(\text{INOUT}) :: \text{sol}\]

\[\text{CHARACTER}(\text{LEN} = 1), \text{INTENT}(\text{IN}), \text{OPTIONAL} :: \text{TRANS}\]

\[\text{REAL}(\text{wp}), \text{INTENT}(\text{OUT}), \text{OPTIONAL} :: \text{err}\]

\[\text{INTEGER}, \text{INTENT}(\text{OUT}), \text{OPTIONAL} :: \text{INFO}\]

where

\[\text{type} ::= \text{REAL} | \text{COMPLEX}\]

\[\text{wp} ::= \text{KIND}(1.0) | \text{KIND}(1.0D0)\]

\[\text{rhs} ::= B(\text{\ldots}) | B(\text{\ldots})\]

\[\text{sol} ::= X(\text{\ldots}) | X(\text{\ldots})\]

\[\text{err} ::= \text{FERR}(:), \text{BERR}(:) | \text{FERR}, \text{BERR}\]

#### A.6.3 Arguments

**A** – *(input)* either **REAL** or **COMPLEX** square array, shape (:, :), \(\text{size}(A, 1) = \text{size}(A, 2)\).

The original matrix \(A\).

**AF** – *(input)* either **REAL** or **COMPLEX** square array, shape (:, :), \(\text{size}(AF, 1) = \text{size}(AF, 2) = \text{size}(A, 1)\). The factors \(L\) and \(U\) from the factorization \(A = PLU\) as computed by **LA_GETRF**.
IPIV – *(input)* INTEGER array, shape (*), \(\text{size}(IPIV) = \text{size}(A, 1)\). The pivot indices from LA_GETRF; for \(1 \leq i \leq \text{size}(A, 1)\), row \(i\) of the matrix was interchanged with row \(IPIV(i)\).

B – *(input)* either REAL or COMPLEX rectangular array, shape either (*; :) or (:), \(\text{size}(B, 1) = \text{size}(A, 1)\). The right hand side vector(s) of matrix \(B\) for the system of equations \(AX = B\).

X – *(input/output)* either REAL or COMPLEX rectangular array, shape either (*; :) or (:), \(\text{size}(X, 1) = \text{size}(A, 1)\).

- 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:
  
  - \(\text{'N'}\) : \(AX = B\) (No transpose)
  - \(\text{'T'}\) : \(A^T X = B\) (Transpose)
  - \(\text{'C'}\) : \(A^H X = B\) (Conjugate transpose)
- otherwise \(TRANS = \text{'N'}\) is assumed.

FERR – *Optional (output)* either REAL array of shape (:), or REAL scalar. If it is an array, \(\text{size}(FERR) = \text{size}(X, 2)\). 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)* either REAL array of shape (:), or REAL scalar. If it is an array, \(\text{size}(BERR) = \text{size}(X, 2)\). 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 LAPACK77 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) either 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.

21
• 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 \(m = \text{size}(A,1)\).

• If $INFO$ is not present and an error occurs, then the program is terminated with an error message.
B  Code for One Version of LA_GESV

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_GESV, with multiple right hand sides (B is a rank-2 array).

B.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.

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

Within the LAPACK90 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:

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

and in double precision:

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

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

B.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_AUX:

```fortran
SUBROUTINE ERINFO(LINFO, SRNAME, INFO)
!  .. Scalar Arguments ..
  CHARACTER( LEN = * ), INTENT(IN) :: SRNAME
  INTEGER , INTENT(IN) :: LINFO
  INTEGER , INTENT(INOUT), OPTIONAL :: INFO
!
!  .. Executable Statements ..
!
  IF( PRESENT(INFO) ) INFO = LINFO
  IF( LINFO < 0 .OR. LINFO>0 .AND. .NOT.PRESENT(INFO) )THEN
```

23
A more elaborate error-handling mechanism could of course be devised.

B.3 Accessing LAPACK routines

We assume that interface-blocks for all the LAPACK routines are accessible from a module LAPACK77_INTERFACES. Note that we do not use generic interfaces for the LAPACK77 routines, since that would impose some restrictions on the way in which LAPACK77 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.

B.4 The code

SUBROUTINE SGESV_F90(A,B,IPIV,INFO)
  ! .. Use Statements ..
  USE LA_PRECISION, ONLY: WP => SP
  USE LA_AUX, ONLY: ERINFO
  USE LAPACK77_INTERFACES, ONLY: GESV_F77 => SGESV
  ! .. Implicit Statement ..
  IMPLICIT NONE
  ! .. Scalar Arguments ..
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  ! .. Array Arguments ..
  INTEGER, INTENT(OUT), OPTIONAL, TARGET :: IPIV(:)
  REAL(WP), INTENT(INOUT) :: A(:,:), B(:,:)
  ! .. Parameters ..
  CHARACTER(LEN=7), PARAMETER :: SRNAME = 'LA_GESV'
  ! .. Local Scalars ..
  INTEGER :: LD, LINFO, NRHS, N
  ! .. Local Pointers ..
  INTEGER, POINTER :: LPIV(:)
  ! .. Intrinsic Functions ..
  INTRINSIC ALLOCATE, DEALLOCATE, MAX, PRESENT, SIZE
  ! .. Executable Statements ..
  !
  ! Test the arguments

WRITE (*,*) 'Program terminated in LAPACK_90 subroutine ', SRNAME
WRITE (*,*) 'Error indicator, INFO = ', LINFO
STOP
END IF
END SUBROUTINE ERINFO
LINFO = 0
N = SIZE(A, 1)
IF( SIZE(A, 2) /= N ) THEN
    LINFO = -1
ELSE IF( SIZE(B, 1) /= N ) THEN
    LINFO = -2
ELSE
    IF( PRESENT(IPIV) ) THEN
        IF( SIZE(IPIV) /= N ) LINFO = -3
    END IF
END IF
END IF

IF( LINFO == 0 ) THEN
    LD = MAX(1, N)
    NRHS = SIZE(B, 2)
    IF( PRESENT(IPIV) ) THEN
        LPIV => IPIV
    ELSE
        ALLOCATE(LPIV(N))
    END IF
    CALL GESV_F77(N, NRHS, A, LD, LPIV, B, LD, LINFO)
END IF

CALL ERINFO(LINFO, SRNAME, INFO)

END SUBROUTINE SGESV_F90