LAPACK FOR FORTRAN90

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

November 25, 1998

Abstract

The purpose of this talk 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 us of assumed-shape arrays, optional arguments, and generic interfaces. 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 a large task. We aim to design an interface which can persist unchanged while the underlying code is rewritten. We aim to maintain the same level of performance as with the Fortran 77 code. In this talk we use as an example the group of LAPACK routines for solving systems of linear equations $AX = B$ with a general matrix A, and for symetric and hermitian eigenproplems, both driver routines and computational routines.

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.

[*]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

[‡]UNI•C, Bldg. 304, Technical University of Denmark, DK-2800 Lyngby, Denmark, Email: jerzy.wasniewski@uni-c.dk

[§]Institute of Mathematics, Polish Academy of Sciences, Śniadeckich 8, 00-950 Warsaw, Poland, Email: adamz@impan.gov.pl

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 designed to allow cross-linking.
2. If such cross-linking is not possible, recompile LAPACK 77 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 LAPACK 77 code to standard Fortran 90 (see below), before recompiling it.

The conversions needed to create standard Fortran 90 code for LAPACK 77 are:

`COMPLEX*16`	$\Rightarrow$	`COMPLEX(KIND=Kind(0.0D0)`
`DCONJG(z)` for `COMPLEX*16 z`	$\Rightarrow$	`CONJG(z)`
`DBLE(z)` for `COMPLEX*16 z`	$\Rightarrow$	`REAL(z)`
`DIMAG(z)` for `COMPLEX*16 z`	$\Rightarrow$	`AIMAG(z)`
`DCMPLX(x,y)` for `DOUBLE PRECISION x, y`	$\Rightarrow$	`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, as illustrated by the following example (in fixed-form source format) containing an interface for the single routine CBDSQR:

```
      MODULE LAPACK77_INTERFACES
      INTERFACE
      SUBROUTINE CBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U,
     $                   LDU, C, LDC, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU
      REAL               D( * ), E( * ), RWORK( * )
      COMPLEX            C( LDC, * ), U( LDU, * ), VT( LDVT, * )
      END
      END INTERFACE
      END MODULE LAPACK77_INTERFACES
```

A single module containing interfaces for all the routines in LAPACK 77 (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 LAPACK 90

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 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 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 specifed 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 acessible 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:

   (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 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: 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 resposibility 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.

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

4 Prototype Implementation of LAPACK 90 Procedures

We have implemented Fortran 90 jacket procedures to the group 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`.

In the appendix of [2], 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 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.

`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 )
```

5 Documentation

In the appendix of [2], 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 `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

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

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. Dongarra, J. Du Croz, S. Hammarling, J. Waśniewski, and A. Zemla. LAPACK Working Note 101, A Proposal for a Fortran 90 Interface for LAPACK. – Report UNIC-95-9, UNI•, Lyngby, Denmark.

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