Testing Software for LAPACK90

Jack Dongarra[*] Wojciech Owczarz[†] Jerzy Waśniewski[†]
Plamen Yalamov[‡]

September 22, 1998

Abstract

LAPACK90 and ScaLAPACK are libraries of high-performance linear algebra subroutines. While LAPACK is developed for scalar, superscalar, and shared memory machines, ScaLAPACK is designed for distributed memory machines. Usually, users are not familiar with details in these subroutines. Therefore, in this paper we describe a possible way to develop testing software for all subroutines in both libraries. For simplicity, we consider only the subroutines for the solution of linear systems. The test subroutines for other linear algebra problems can be developed in a similar way. The test programs are written in FORTRAN90, and we use LAPACK90, which interfaces FORTRAN90 with LAPACK.

1 Introduction

The high performance packages LAPACK [0] and ScaLAPACK [0] are powerful tools for solving linear algebra problems. New standards of FORTRAN have been defined; FORTRAN90 and HPF (High Performance FORTRAN). Interface libraries between FORTRAN90 and LAPACK (LAPACK90), and between HPF and ScaLAPACK, have been developed. In this paper, we briefly introduce these packages and then propose a way to test all the subroutines in them. All of the test programs will be put together as a subdirectory with the next release of the LAPACK90 library.

1.1 FORTRAN 90

FORTRAN has always been the principal language used in the fields of scientific, numerical, and engineering computing. A series of revisions to the standard defining successive versions of the language has progressively enhanced its power and kept it competitive with several generations of rivals. The present FORTRAN standard is 90/95. A summary of the new features follows:

[*]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; e-mail: dongarra@cs.utk.edu

[†]The Danish Computing Centre for Research and Education (UNI•C), Technical University of Denmark, Building 304, DK-2800 Lyngby, Denmark, e-mail: uniwow@uni-c.dk, or e-mail: jerzy.wasniewski@uni-c.dk

[‡]Center of Applied Mathematics and Informatics, University of Rousse, 7017 Rousse, Bulgaria, e-mail: yalamov@ami.ru.acad.bg. This author was supported in part by Grant I-702/97 from the National Scientific Research Fund of the Bulgarian Ministry of Education and Science.

- Array operations.
- Pointers.
- Improved facilities for numerical computations including a set of numerical inquiry functions.
- Parameterization of the intrinsic types, to permit processors to support short integers, very large character sets, more than two precisions for real and complex, and packed logicals.
- User-defined derived data types composed of arbitrary data structures and operations upon those structures.
- Facilities for defining collections called "modules", useful for global data definitions and for procedure libraries. These support a safe method of encapsulating derived data types.
- Requirements on a compiler to detect the use of constructs that do not conform to the syntax of the language or are obsolete.
- A new source form, more appropriate to use at a terminal.
- New control constructs such as the SELECT CASE construct and a new form of the DO.
- The ability to write internal procedures and recursive procedures, and to call procedures with optional and keyword arguments.
- Dynamic storage (automatic arrays, allocatable arrays, and pointers).
- Improvements to the input-output facilities, including handling partial records and a standardized NAMELIST facility.
- Many new intrinsic procedures.

Together, the new features contained in FORTRAN 90/95 ensure that the FORTRAN language will continue to be used successfully for a long time to come. The fact that it contains the whole of FORTRAN 77 as a subset means that conversion to FORTRAN 90/95 is as simple as conversion to another FORTRAN 77 processor. For more information on FORTRAN 90/95 see [0].

1.2 LAPACK

LAPACK is a library of FORTRAN 77 subroutines for solving the most commonly occurring problems in numerical linear algebra. It has been designed for efficiency on a wide range of modern, high-performance computers. The name LAPACK is an acronym for Linear Algebra PACKage.

LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. These block operations can be optimized for each architecture to account for the memory hierarchy, and so provide a transportable way to achieve high efficiency on diverse modern machines. LAPACK requires that highly optimized block matrix operations are already implemented on each machine.

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms [0] (BLAS). While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS, LAPACK is designed at the outset to exploit the Level 3 BLAS – a set of specifications for FORTRAN subprograms that does various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides. Because of the coarse granularity of the Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers, particularly if specially coded implementations are provided by the manufacturer.

Highly efficient, machine-specific implementations of the BLAS are available for many modern, high-performance computers. The BLAS enable LAPACK routines to achieve high performance with transportable software. It is not expected to perform as well as a specially tuned implementation on most high-performance computers. On some machines it may give much worse performance. But it allows users to run LAPACK software on machines that do not offer any other implementation of the BLAS. A model FORTRAN implementation of the BLAS is available from netlib [0] in the BLAS library.

For more information on LAPACK and references on BLAS, LINPACK and EISPACK see [0, 0].

1.3 LAPACK for FORTRAN 90

All LAPACK driver subroutines (including expert drivers) and some LAPACK computationals have both generic LAPACK90 interfaces and generic LAPACK77 interfaces. The remaining computationals have only generic LAPACK77 interfaces. In both types of interfaces, no distinction is made between single and double precision or between real and complex data types. The use of the LAPACK90 (LAPACK77) interface requires the user to specify the F90_LAPACK (F77_LAPACK) module.

For example, the GESV driver subroutine, which solves a general system of linear equations, can be called in the following ways:

- CALL LA_GESV(A, B, IPIV=ipiv, INFO=info)
  or
- CALL LA_GESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)

The module F90_LAPACK is needed in the first case in which the LAPACK90 interface package is called. The module F77_LAPACK is needed in the second case in which the LAPACK77 package is directly called.

The implementation of the LAPACK90 can be summarized as follows:

- Driver Routines for Linear Equations.
- Expert Driver Routines for Linear Equations.
- Driver Routines for Linear Least Squares Problems.
- Driver Routines for generalized Linear Least Squares Problems.
- Driver Routines for Standard Eigenvalue and Singular Value Problems.
- Divide and Conquer Driver Routines for Standard Eigenvalue Problems.
- Expert Driver Routines for Standard Eigenvalue Problems.
- Driver Routines for Generalized Eigenvalue and Singular Value Problems.
- Some Computational Routines for Linear Equations and Eigenproblems.

The LAPACK90 library is successively updated and is available from netlib (see [0, 0]).

1.4 ScaLAPACK

ScaLAPACK is a library of high-performance linear algebra routines for distributed memory message-passing MIMD (Multiple Instruction Multiple Data) computers and networks of workstations supporting PVM [0] (Parallel Virtual Machine) and/or MPI [0] (Message Passing Interface). ScaLAPACK is a continuation of the LAPACK project (see section). Both libraries (LAPACK and ScaLAPACK) contain routines for solving systems of linear equations, least squares problems, and eigenvalue problems. The goals of both projects are efficiency (to run as fast as possible), scalability (as the problem size and number of processors grow), reliability (including error bounds), portability (across all important parallel machines), flexibility (so users can construct new routines from well-designed parts), and ease of use (by making the interface to LAPACK and ScaLAPACK look as similar as possible). Many of these goals, particularly portability, are aided by the development and promotion of standards, especially for low-level communication and computation routines. ScaLAPACK has been successful in attaining these goals, limiting most machine dependencies to two standard libraries called the BLAS (Basic Linear Algebra Subprograms) and BLACS [0] (Basic Linear Algebra Communication Subprograms). LAPACK runs on any machine where the BLAS [0] are available, and ScaLAPACK runs on any machine where both the BLAS and the BLACS are available.

The library is currently written in FORTRAN 77 (with the exception of a few symmetric eigenproblem auxiliary routines written in C to exploit IEEE arithmetic) in a Single Program Multiple Data (SPMD) style using explicit message passing for interprocessor communication. The name ScaLAPACK is an acronym for Scalable Linear Algebra PACKage, or Scalable LAPACK

For more information on ScaLAPACK and references on BLAS, BLACS, PBLAS, PVM and MPI see [0, 0, 0, 0, 0, 0].

1.5 ScaLAPACK for HPF

Work on the HPF interface for ScaLAPACK has been started by a number of groups, such as the University of Tennessee and the Danish Computing Center for Research and Education (UNI•C) (see [0, 0]). The plan is to develop an HPF interface for several of the more heavily used ScaLAPACK subroutines and test programs.

2 Testing routines for LAPACK90

We will present the testing routine for SGESV (solution of linear systems with general dense matrices). The rest of the testing routines are similar, and we give one more (for banded matrices) in the Appendix.

The naming convention for the testing routines is as follows. We add the letters MG (Matrix Generator) at the end of the corresponding subroutine name. For example, the testing routine for SGESV is called SGESVMG.

Now let us present the details for SGESVMG. The code starts with statements for the variables.

```
      PROGRAM LA_SGESV_MG_EXAMPLE
!
!  -- LAPACK90 Testing Routine (VERSION 1.0) --
!     Danish Computing Center (UNI-C), Denmark
!     University of Rousse, Bulgaria
!     University of Tennessee, USA
!     Aug 5, 1998
!
!     .. "Use Statements" ..
      USE LA_PRECISION, ONLY: WP => SP
      USE F90_LAPACK, ONLY: LA_GESV, LA_LAGGE, LA_GETRF
!     .. "Implicit Statement" ..
      IMPLICIT NONE
!     .. "Parameters" ..
      INTEGER, PARAMETER :: NSTART = 50, NINCR = 20, NSTOP = 100, &
               NRHS = 50, NIN = 5, NOUT = 6, NTESTS = 4, &
                           NETESTS = 9
      REAL(WP), PARAMETER :: THRESH_FAC = 100.0
!     THRESH IS EQUAL TO THE PRODUCT OF THRESH_FAC AND EPS
!                      (THE MACHINE EPSILON)
      REAL(WP) :: THRESH
!     .. "Local Scalars" ..
      INTEGER :: FETESTS, FMATR, FTESTS, INFO, ISEED(4), ISTAT, J, N, NMATR
      REAL(WP) :: EPS, RATIO, RCOND
!     .. "Local Arrays" ..
      INTEGER, ALLOCATABLE :: IPIV(:)
      REAL(WP), ALLOCATABLE :: A(:,:), AA(:,:), B(:,:), BB(:,:), DUMMY(:,:)
      REAL(WP), ALLOCATABLE :: D(:)
```

Here we explain the most important of the parameters. We do tests with different sizes of the matrix. The size grows from NSTART to NSTOP with a step NINCR. For each size, we repeat all the tests in a DO-loop. In some tests, we solve the problem with multiple right-hand sides. The corresponding number is given in NRHS. Finally, we introduce a threshold for the accuracy. The variable THRESH_FAC is a factor by which the machine precision is multiplied, so that the product is a threshold for the componentwise backward error. As we will see later, if the componentwise backward error is above this threshold we produce an error message. Since this example program (SGESV) is in single precision (machine precision $\approx 10^{-7}$), we give the value of 100 to THRESH_FAC. Thus, backward errors larger than $\approx 10^{-5}$ will be reported to the user as a possible danger. Gaussian elimination with partial pivoting (which is implemented in SGESV) is quite stable in practice, so we would rarely expect larger than the threshold errors.

Next we give values to some counters and messages explaining the routine:

```
!      .. "Executable Statements" ..
       FTESTS = 0; FETESTS = 0; NMATR = 0; FMATR = 0

       WRITE(NOUT,*)
       WRITE (NOUT,*) 'SGESV Test Example Program Results.'
       WRITE(NOUT,*) 'LA_GESV LAPACK subroutine solves a dense general'
       WRITE(NOUT,*) 'linear system of equations, Ax = b.'
       EPS = EPSILON(1.0_WP)
       THRESH = THRESH_FAC * EPS
       WRITE(NOUT,'( 1X, A, E12.5 )') 'Threshold value for the backward error &
                    = ',THRESH
       WRITE(NOUT,'( 1X, A, E12.5 )') 'The machine eps = ', EPS
```

We start a DO-loop making tests for different sizes of the matrices. Memory is allocated for all arrays, and a message is produced if the memory is not enough:

```
!
       DO N = NSTART, NSTOP, NINCR
       NMATR = NMATR + 1
!
       ALLOCATE ( A(N,N), AA(N,N), B(N,NRHS), BB(N,NRHS), IPIV(N), D(N), &
                    STAT=ISTAT )
       IF( ISTAT /= 0 )THEN
          WRITE(NOUT,*) 'Program can not allocate more memory, STA = ', ISTAT
          STOP
       END IF
```

A random matrix and a random block of right-hand sides are generated in AA and BB. The reciprocal of a condition number of the matrix is then estimated, and the linear system is solved:

```
! GENERATE A MATRIX
       CALL LA_LAGGE( AA )

! GENERATE RHS
```

```
      CALL LA_LAGGE( BB )

! CALCULATE THE CONDITION NUMBER OF THE MATRIX AA
      A = AA
      CALL LA_GETRF(A, RCOND=RCOND)

! CALL THE SOLVER
      A=AA; B=BB
      CALL LA_GESV(  A, B, IPIV, INFO )
```

After that we compute the componentwise backward error of the solution by a call to CWBE. We discuss this subroutine at the end. The backward error is stored in RATIO.

```
! COMPUTE  THE COMPONENTWISE BACKWARD ERROR

      CALL CWBE(AA, B, BB, RATIO)
```

We produce an error message if INFO is not zero (the termination of GESV is not normal), or the backward error is too large (which means that the solution is not accurate enough). We report the value of INFO (which shows the type of error in GESV) and the values of RCOND and RATIO (which show the conditioning and the backward error for the linear system) so that the user can make a conclusion.

```
      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 1 -- ''CALL LA_GESV( A, B, IPIV, INFO )'',', &
                       'Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', &
                  N, ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE MAXIMAL COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
```

This was the first type of test presented in detail. Next we repeat the same test but for a linear system with one right-hand side only:

```
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(:,1), IPIV, INFO )
      CALL CWBE(AA, B(:,1:1), BB(:,1:1), RATIO)
```

```
! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 2 -- ''CALL LA_GESV( A, B(:,1), IPIV, INFO )'', &
                        Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', &
                    N, ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
```

A new random matrix is then generated, and the last two tests are done for the newly obtained linear system (the right hand sides remain the same):

```
!
      ISEED(1)=4000; ISEED(2)=50; ISEED(3)=1997; ISEED(4)=11
      CALL LA_LAGGE( AA, ISEED=ISEED )
      CALL LA_GETRF(AA, RCOND=RCOND)
      A=AA; B=BB
      CALL LA_GESV(  A, B )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B, BB, RATIO)

! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 3 -- ''CALL LA_GESV( A, B )'', Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                     ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
```

```
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*) 'THE MAXIMAL COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(:,1) )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B(:,1:1), BB(:,1:1), RATIO)

! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 4 -- ''CALL LA_GESV( A, B(:,1) )'', Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                     ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*) 'THE COMPONENTWISE BACKWARD ERROR IS:',  RATIO
      END IF
```

Finally, we close the DO-loop (which changes the matrix size) and produce a report for all the tests up to this point:

```
!
      DEALLOCATE ( A, AA, B, BB, IPIV, D, STAT=ISTAT )
!
      END DO
!
      WRITE(NOUT,*)
      WRITE(NOUT,*)'---------------------------------------------------------'
      WRITE(NOUT,*)
      WRITE(NOUT,'( I4, A, I2, A, I4, A )') NMATR, ' matrices were tested', &
                 ' with ', NTESTS, ' tests. NRHS was ', NRHS, ' and one.'
      WRITE(NOUT,'( I4, A )') NMATR*NTESTS - FTESTS, ' tests passed.'
      WRITE(NOUT,'( I4, A )') FTESTS, ' tests failed.'
```

The second part of the test focuses on the tests for the arguments of SGESV. Pointing to the DUMMY array, which is not allocated in the memory, tests the first argument. In this case, INFO should be negative:

```
!     TESTS FOR THE ARGUMENT ERROR FAULTS
      N = 100
      ALLOCATE ( A(N,N), AA(N,N), B(N,NRHS), BB(N,NRHS), IPIV(N), D(N) )
!
      A=AA; B=BB
      CALL LA_GESV(  DUMMY, B, INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( DUMMY, B, INFO=INFO )'' failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  DUMMY, B(:,1), INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( DUMMY, B(:,1), INFO=INFO )'' &
                       failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
```

The arrays A and B are then "cut" in different ways. In this case, INFO should also be negative:

```
!
      A=AA; B=BB
      CALL LA_GESV(  A(:,1:N-1), B, INFO=INFO )
      IF( INFO /= -1 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A(:,1:N-1), B, INFO=INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -1'
      END IF
!
      A=AA; B=BB
```

```
      CALL LA_GESV(  A(:,1:N-1), B(:,1), INFO=INFO )
      IF( INFO /= -1 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A(:,1:N-1), B(:,1), ', &
                       ' INFO=INFO )'' failed,'
         WRITE(NOUT,*) 'INFO returned should be -1'
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(1:N-1,:), INFO=INFO )
      IF( INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A, B(1:N-1,:), INFO=INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -2'
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(1:N-1,1), INFO=INFO )
      IF( INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A, B(1:N-1,1), INFO=INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -2'
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(1:N-1,:), INFO=INFO )
      IF( INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
```

```
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A, B(1:N-1,:), INFO=INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -2'
      END IF
```

The same type of test is applied to IPIV:

```
!
      A=AA; B=BB
      CALL LA_GESV(  A, B, IPIV(1:N-1), INFO )
      IF( INFO /= -3 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A, B, IPIV(1:N-1), INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -3'
      END IF
!
      A=AA; B=BB
      CALL LA_GESV(  A, B(:,1), IPIV(1:N-1), INFO )
      IF( INFO /= -3 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GESV( A, B, IPIV(1:N-1), INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be -3'
      END IF
```

At the end, we provide a report for all the tests:

```
!
      WRITE(NOUT,*)
      WRITE(NOUT,*)'-----------------------------------------------------'
      WRITE(NOUT,*)
      WRITE(NOUT,'( I2, A )') NETESTS, ' error exits tests were ran'
      WRITE(NOUT,'( I4, A )') NETESTS - FETESTS, ' tests passed.'
      WRITE(NOUT,'( I4, A )') FETESTS, ' tests failed.'
!
      CONTAINS
```

The subroutine CWBE, which is called inside the test program, is given as follows:

```
      SUBROUTINE CWBE(AA, X, B, RATIO)
      USE LA_PRECISION, ONLY: WP => SP
      IMPLICIT NONE
      REAL(WP), INTENT(IN) :: AA(:,:), X(:,:), B(:,:)
      REAL(WP), INTENT(OUT) :: RATIO

      INTEGER :: J
      INTRINSIC SIZE
! COMPUTE  THE COMPONENTWISE BACKWARD ERROR

      RATIO = 0.0_WP
      DO J = 1, SIZE(B,2)
        RATIO = MAX(RATIO, MAXVAL((ABS(B(:,J) - MATMUL(AA,X(:,J)))) / &
        (ABS(X(:,J)) + MATMUL(ABS(AA),ABS(X(:,J))))))
      END DO
      END SUBROUTINE
```

To compute the componentwise backward error, we use the expression (see [0])

$$\omega_c = \max_i \frac{|r_i|}{(|A||\hat{x}| + |b|)_i},$$

where $r = b - A\hat{x}$ is the residual, and $\hat{x}$ is the computed solution. This is the case of one right-hand side. When we have multiple right-hand sides, we take the maximum of all the componentwise backward errors.

The output of the test program appears as follows:

```
 SGESV Test Example Program Results.
 LA_GESV LAPACK subroutine solves a dense general
 linear system of equations, Ax = b.
 Threshold value for the backward error =  0.11921E-04
 The machine eps =  0.11921E-06

 --------------------------------------------------------

   3 matrices were tested with  4 tests. NRHS was   50 and one.
  12 tests passed.
   0 tests failed.

 --------------------------------------------------------

 9 error exits tests were ran
   9 tests passed.
   0 tests failed.
```

3 A testing routine for ScaLAPACK

The testing routines for ScaLAPACK are designed in the same way, so we do not present them here. If HPF is used, the tests must be updated with the following HPF directives:

```
!HPF$ PROCESSORS PP(NPP,NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE (CYCLIC(MB),CYCLIC(NB)) ONTO:: AA, BB, A, B
!HPF$ ALIGN IPIV(I) WITH A(I,*)
```

where PP is the name of the processors, MB and NB are the block sizes, and AA, BB, A, B and PIV are arrays used in the program.

Our test programs can also be used directly with ScaLAPACK. The data communication can be done using BLACS [0], as in this case.

The test programs can also be used for testing the computer speed. Of course, the machine dependent timing routine must update them.

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] L.S. Blackford, J. Choi, A. Ceary, E. D'Azevedo, J. Demmel, I. Dhilon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, 1997.

[3] L.S. Blackford, J.J. Dongarra, J. Du Croz, S. Hammarling, and J. Waśniewski. *LAPACK90 - FORTRAN90 version of LAPACK.* On web: http://www.netlib.org/lapack90/ (1997)

[4] L.S Blackford, J.J. Dongarra, J. Du Croz, S. Hammarling, and J. Waśniewski. *LAPACK Working Note 117, A Proposal for a FORTRAN 90 Interface for LAPACK.* Report UNIC-96-10, UNI•C, Lyngby, Denmark, 1995. Report ut-cs-96-341, University of Tennessee, Computer Science Department, Knoxville, July, 1995.

[5] *BLACS (Basic Linear Algebra Communication Subprograms).* See at netlib http://www.cs.utk.edu/r̃whaley/Blacs.html

[6] *BLAS (Basic Linear Algebra Subprograms).* See at netlib http://www.netlib.org/blas/index.html

[7] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. *The Netlib Mathematical Software Repository.* D-Lib Magazine, Sep, 1995, Accessible at http://www.dlib.org/

[8] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and, R. C. Whaley. *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms.* University of Tennessee at Knoxville, Technical Report, CS-95-292, May 1995. Accessible at http://www.netlib.org/lapack/lawns/index.html (lapack/lawns/lawn100.ps).

[9] J. Dongarra and J. Wasniewski, High Performance Linear Algebra Package LAPACK90, Report UNIC-98-01, February 1998.

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam *PVM: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[11] C.H. Koelbel, D.B. Lovemann, R.S. Schreiber, G.L. Steele Jr., and M.E. Zosel. *The High Performance FORTRAN Handbook.* The MIT Press Cambridge, Massachusetts, London, England, 1994.

[12] P.A.R. Lorenzo, A. Müller, Y. Murakami, and B.J.N. Wylie. High Performance FORTRAN Interfacing to ScaLAPACK. In J. Waśniewski, J. Dongarra, K. Madsen, and D. Olesen (Eds.), Applied Parallel Computing, Industrial Computation and Optimization, Third International Workshop, PARA'96, Lyngby, Denmark, August 1996, Proceedings, Lecture Notes in Computer Science No. 1184, Springer-Verlag, 1996, pp. 457-466

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

[14] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference.* The MIT Press Cambridge, Massachusetts, 1996.

[15] *R.C. Whaley. HPF Interface to ScaLAPACK.*
On web: http://www.netlib.org/scalapack/prototype/ (1997).

A SGBSVMG

```
      PROGRAM LA_SGBSV_MG_EXAMPLE
!
!  -- LAPACK90 Testing Routine (VERSION 1.0) --
!     Danish Computing Center (UNI-C), Denmark
!     University of Rousse, Bulgaria
!     University of Tennessee, USA
!     Aug 5, 1998
!
!     .. "Use Statements" ..
      USE LA_PRECISION, ONLY: WP => SP
      USE F90_LAPACK, ONLY: LA_GBSV, LA_LAGGE, LA_GETRF
!     .. "Implicit Statement" ..
      IMPLICIT NONE
!     .. "Parameters" ..
      INTEGER, PARAMETER :: NSTART = 50, NINCR = 20, NSTOP = 100, &
                            NRHS = 50, NIN = 5, NOUT = 6, NTESTS = 6, &
                            NETESTS = 8
      REAL(WP), PARAMETER :: THRESH_FAC = 100.0
!     THRESH IS EQUAL TO THE PRODUCT OF THRESH_FAC AND EPS
```

```
!     (THE MACHINE EPSILON)
      REAL(WP) :: THRESH
!     .. "Local Scalars" ..
      INTEGER :: FETESTS, FMATR, FTESTS, INFO, ISTAT, J, N, NMATR, KL, KU
      REAL(WP) :: EPS, RATIO, RCOND
!     .. "Local Arrays" ..
      REAL(WP), ALLOCATABLE :: AB(:,:), AAB(:,:), A(:,:), AA(:,:)
      REAL(WP), ALLOCATABLE :: B(:,:), BB(:,:), DUMMY(:,:)
      INTEGER,  ALLOCATABLE :: IPIV(:)
!     .. "Executable Statements" ..
      FTESTS = 0; FETESTS = 0; NMATR = 0; FMATR = 0

      WRITE(NOUT,*)
      WRITE (NOUT,*) 'SGBSV Test Example Program Results.'
      WRITE(NOUT,*) 'LA_GBSV LAPACK subroutine solves a system of linear'
      WRITE(NOUT,*) 'equations Ax = b, where A is banded. '

      EPS = EPSILON(1.0_WP)
      THRESH = THRESH_FAC * EPS
      WRITE(NOUT,'( 1X, A, E12.5 )') 'Threshold value for the backward', &
                                     ' error = ',THRESH
      WRITE(NOUT,'( 1X, A, E12.5 )') 'The machine eps = ', EPS
!
      DO N = NSTART, NSTOP, NINCR
      NMATR = NMATR + 1
!
      ALLOCATE( A(N,N), AA(N,N), AB(N,N), AAB(N,N), &
                B(N,NRHS), BB(N,NRHS), STAT=ISTAT )
      IF( ISTAT /= 0 )THEN
        WRITE(NOUT,*) 'Program can not allocate more memory, STAT = ', ISTAT
        STOP
      END IF

      KL = (N-1)/2 ; KU = N-2*KL - 1
! NEXT SUBROUTINE GENERATES DENSE MATRIX AA WITH KL SUBDIAGONALS AND KU
! SUPERDIAGONALS
! THE RHS IS STORED IN BB. THE RECIPROCAL OF THE CONDITION NUMBER FOR AA IS
    ! RETURNED IN RCOND
      CALL GENERATEMATRICES(AA, AB, BB, KL, KU, RCOND)

!
! CALL THE SOLVER
      AB=AAB; B=BB
      CALL LA_GBSV(AB, B, KL, INFO=INFO )
```

```
! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B, BB, RATIO)

! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 1 -- ''CALL LA_GBSV( AB, B, KL, INFO )'', Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                         ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE MAXIMAL COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV( AB, B(:,1), KL, INFO=INFO )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B(:,1:1), BB(:,1:1), RATIO)

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 2 -- ''CALL LA_GBSV( AB, B(:,1), KL, INFO )'', &
                      Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                      ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF

      KL = (N-1)/4; KU = N-2*KL - 1
```

```
! NEXT SUBROUTINE GENERATES DENSE MATRIX AA WITH KL SUBDIAGONALS AND KU
! SUPERDIAGONALS
! THE RHS IS STORED IN BB. THE RECIPROCAL OF THE CONDITION NUMBER FOR AA IS
   ! RETURNED IN RCOND
      CALL GENERATEMATRICES(AA, AB, BB, KL, KU, RCOND)

! CALL THE SOLVER
      AB=AAB; B=BB
      CALL LA_GBSV(AB, B, KL, INFO=INFO )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B, BB, RATIO)

! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
        FTESTS = FTESTS + 1
        FMATR = FMATR + 1
        WRITE(NOUT,*)
        WRITE(NOUT,*)'-----------------------------------------------------'
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'Test 3 -- ''CALL LA_GBSV( AB, B, KL, INFO )'', Failed.'
        WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                      ' with ', NRHS, ' rhs.'
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'INFO = ', INFO
        WRITE(NOUT,*) ' RCOND = ', RCOND
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'THE MAXIMAL COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV( AB, B(:,1), KL, INFO=INFO )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B(:,1:1), BB(:,1:1), RATIO)

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
        FTESTS = FTESTS + 1
        FMATR = FMATR + 1
        WRITE(NOUT,*)
        WRITE(NOUT,*)'-----------------------------------------------------'
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'Test 4 -- ''CALL LA_GBSV( AB, B(:,1), KL, INFO )'', &
                       Failed.'
```

```
        WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                      ' with ', NRHS, ' rhs.'
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'INFO = ', INFO
        WRITE(NOUT,*) ' RCOND = ', RCOND
        WRITE(NOUT,*)
        WRITE(NOUT,*) 'THE COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF

      KL = 0 ; KU = N-2*KL - 1
! NEXT SUBROUTINE GENERATES DENSE MATRIX AA WITH KL SUBDIAGONALS AND KU
! SUPERDIAGONALS
! THE RHS IS STORED IN BB. THE RECIPROCAL OF THE CONDITION NUMBER FOR AA IS
    ! RETURNED IN RCOND
      CALL GENERATEMATRICES(AA, AB, BB, KL, KU, RCOND)

! CALL THE SOLVER
      AB=AAB; B=BB
      CALL LA_GBSV(AB, B, KL, INFO=INFO )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
      CALL CWBE(AA, B, BB, RATIO)

! THE COMPONENTWISE BACKWARD ERROR IS IN RATIO

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 5 -- ''CALL LA_GBSV( AB, B, KL, INFO )'', Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                      ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE MAXIMAL COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV( AB, B(:,1), KL, INFO=INFO )

! COMPUTE  THE COMPONENTWISE BACKWARD ERROR
```

```
      CALL CWBE(AA, B(:,1:1), BB(:,1:1), RATIO)

      IF( INFO /= 0 .OR. RATIO > THRESH )THEN
         FTESTS = FTESTS + 1
         FMATR = FMATR + 1
         WRITE(NOUT,*)
         WRITE(NOUT,*)'----------------------------------------------------'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'Test 6 -- ''CALL LA_GBSV( AB, B(:,1), KL, INFO )'', &
                        Failed.'
         WRITE(NOUT,'( A, I4, A, I4, A, I4, A )') 'Matrix ', N, ' x', N, &
                       ' with ', NRHS, ' rhs.'
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) ' RCOND = ', RCOND
         WRITE(NOUT,*)
         WRITE(NOUT,*) 'THE COMPONENTWISE BACKWARD ERROR IS: ',  RATIO
      END IF

      DEALLOCATE( A, AA, AB, AAB, B, BB, STAT=ISTAT )
      END DO
!
      WRITE(NOUT,*)
      WRITE(NOUT,*)'-------------------------------------------------------'
      WRITE(NOUT,*)
      WRITE(NOUT,'( I4, A, I2, A, I4, A )') NMATR, ' matrices were tested', &
                 ' with ', NTESTS, ' tests. NRHS was ', NRHS, ' and one.'
      WRITE(NOUT,'( I4, A )') NMATR*NTESTS - FTESTS, ' tests passed.'
      WRITE(NOUT,'( I4, A )') FTESTS, ' tests failed.'
!
!     TESTS FOR THE ARGUMENT ERROR FAULTS
      N = 100
      ALLOCATE ( AB(N/2,N), AAB(N/2,N), B(N,NRHS), BB(N,NRHS), IPIV(N) )
!
      AB=AAB; B=BB
      CALL LA_GBSV( DUMMY, B, INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( DUMMY, B, INFO=INFO )'' failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
```

```
!
      AB=AAB; B=BB
      CALL LA_GBSV(  DUMMY, B(:,1), INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( DUMMY, B(:,1), INFO=INFO )'' &
                        failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV(  AB(:,1:N-1), B, INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( AB(:,1:N-1), B,', &
                       ' INFO=INFO )'' failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV(  AB(:,1:N-1), B(:,1), INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -2)THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( AB(:,1:N-1), B(:,1),', &
                       ' INFO=INFO )'' failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -2'
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV(  AB, B(1:N-1,:), INFO=INFO )
      IF( INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
```

```
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( AB, B(1:N-1,:), INFO=INFO )'' &
                       failed,'
         WRITE(NOUT,*) 'INFO returned should be -2'
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV( AB, B(1:N-1,1), INFO=INFO )
      IF( INFO /= -2 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( AB, B(1:N-1,1), INFO=INFO )'' &
                       failed,'
         WRITE(NOUT,*) 'INFO returned should be -2'
      END IF
!
      AB=AAB; B=BB
      CALL LA_GBSV(  AB, B, (N-1)/2 + 1, INFO=INFO )
      IF( INFO /= -1 .AND. INFO /= -3 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( A, B, (N-1)/2 + 1, INFO )'' &
                       failed,'
         WRITE(NOUT,*) 'INFO returned should be either -1 or -3'
       END IF

       AB=AAB; B=BB
       CALL LA_GBSV(  AB, B, IPIV=IPIV(1:N-1), INFO=INFO )
       IF( INFO /= -4 )THEN
         WRITE(NOUT,*)
         WRITE(NOUT,*)'-----------------------------------------------------'
         WRITE(NOUT,*)
         FETESTS = FETESTS +1
         WRITE(NOUT,*) 'INFO = ', INFO
         WRITE(NOUT,*) 'Test ''CALL LA_GBSV( A, B, IPIV=IPIV, INFO=INFO )'' &
                       failed,'
         WRITE(NOUT,*) 'INFO returned should be -4'
```

```
       END IF
!
      WRITE(NOUT,*)
      WRITE(NOUT,*)'-----------------------------------------------------'
      WRITE(NOUT,*)
      WRITE(NOUT,'( I2, A )') NETESTS, ' error exits tests were ran'
      WRITE(NOUT,'( I4, A )') NETESTS - FETESTS, ' tests passed.'
      WRITE(NOUT,'( I4, A )') FETESTS, ' tests failed.'
!
      CONTAINS

      SUBROUTINE GENERATEMATRICES(AA, AB, BB, KL, KU, RCOND )
      USE LA_PRECISION, ONLY: WP => SP
      USE F90_LAPACK, ONLY: LA_LAGGE, LA_GETRF
      REAL(WP), INTENT(OUT) :: AA(:,:), AB(:,:), BB(:,:), RCOND
      INTEGER, INTENT(IN) :: KL, KU
      INTEGER :: I, J, N, NRHS

!     .. "Implicit Statement" ..
      IMPLICIT NONE
      INTRINSIC MIN, MAX, SUM

      N = SIZE(A,1)
      NRHS = SIZE(B,2)
!
! GENERATE A MATRIX
      CALL LA_LAGGE(AA, KL, KU)
      DO I = 1, N
        IF (AA(I,I) == 0) THEN
          AA(I,I) = I
        END IF
      END DO
!
! GENERATE RHS
      CALL LA_LAGGE( BB )
!
! CALCULATE THE RECIPROCAL OF THE CONDITION NUMBER OF MATRIX AA
      A = AA
      CALL LA_GETRF(A, RCOND=RCOND)
!
! STORE IT AS BAND MATRIX AS NEED BY LA_SGBSV
      DO J=1, N
        DO I=MAX(1, J-KU), MIN(N,J+KL)
          AAB(KL+KU+1+I-J:KL+KU+1+I-J, J) = AA(I,J)
        END DO
```

```
      END DO
      END SUBROUTINE

      SUBROUTINE CWBE(AA, X, B, RATIO)
      USE LA_PRECISION, ONLY: WP => SP
      IMPLICIT NONE
      REAL(WP), INTENT(IN) :: AA(:,:), X(:,:), B(:,:)
      REAL(WP), INTENT(OUT) :: RATIO

      INTEGER :: J
      INTRINSIC SIZE
! COMPUTE  THE COMPONENTWISE BACKWARD ERROR

      RATIO = 0.0_WP
      DO J = 1, SIZE(B,2)
        RATIO = MAX(RATIO, MAXVAL((ABS(B(:,J) - MATMUL(AA,X(:,J)))) / &
        (ABS(X(:,J)) + MATMUL(ABS(AA),ABS(X(:,J))))))
      END DO
      END SUBROUTINE

      END PROGRAM LA_SGBSV_MG_EXAMPLE
```