Source file: `tdgesv.f`

```
c===========================================================
c     Test program for LAPACK "driver" routine 'dgesv'
c     which computes the solution of a real system
c     of linear equations:  A x = b
c
c     This version uses fixed-size 2-d arrays (size fixed at
c     some maximum value commensurate with needs and/or
c     available memory), illustrating another commonly used
c     Fortran technique to implement run-time dimensioning,
c     PARTICULARLY FOR RANK-2 ARRAYS.
c
c     This time the rules are as follows:  All subroutines and
c     functions which manipulate the array must be passed:
c
c     (1)  The array itself.
c     (2)  The "true" or "physical" dimensions;
c          i.e. the dimensions in MAIN (*).
c     (3)  The "run-time" or "logical" dimensions (*).
c
c     (*)  More precisely, due to the nature of the FORTRAN
c          subscripting computation, the leading d-1 dimensions
c          must be passed for a rank-d array.  In particular,
c          for rank-2 array (matrices), THE leading physical
c          dimension (often denoted 'LDA' in LAPACK code), and
c          THE leading logical dimension (often denoted 'N')
c          must BOTH be passed.
c
c     Passing the physical dimensions ensures that the
c     linearization/subscripting calculation is identical
c     in all program units INCUDING MAIN---so that, e.g.,
c     one can safely and conveniently use a(i,j) etc. in
c     MAIN.
c
c     Passing the logical dimensions allows us to write
c     routines which function for a general case (here,
c     typically for N x N matrices).
c
c     Passing BOTH sets of dimensions is slightly cumbersome,
c     but is the price we pay in this case for convenience
c     and generality.
c===========================================================
      program         tdgesv

      implicit        none
c-----------------------------------------------------------
c     Maximum size of linear system.
c-----------------------------------------------------------
      integer         maxn
      parameter     ( maxn = 100 )

c-----------------------------------------------------------
c     Storage for arrays.
c-----------------------------------------------------------
      real*8          a(maxn,maxn),
     &                b(maxn)
      integer         ipiv(maxn)

      integer         i,              nrhs,
     &                n,              info

c-----------------------------------------------------------
c     Set up sample 3 x 3 system ...
c-----------------------------------------------------------
      a(1,1) =  1.23d0
      a(1,2) =  0.24d0
      a(1,3) = -0.45d0

      a(2,1) = -0.43d0
      a(2,2) =  2.45d0
      a(2,3) =  0.78d0

      a(3,1) =  0.51d0
      a(3,2) = -0.68d0
      a(3,3) =  3.23d0

      b(1)   =  6.78d0
      b(2)   = -3.45d0
      b(3)   =  1.67d0
c-----------------------------------------------------------
c     ... and solve it.  Note that 'dgsev' is general
c     enough to solve A x_i = b_i for multiple right-hand-
c     sides b_i.  Here we have only one right-hand-side.
c     Also note that the procedure performs the LU
c     decomposition in place, thus destroying the
c     input-matrix, it also overwrites the right-hand-side(s)
c     with the solution(s).  Finally, observe that we
c     pass the "leading dimension" (maxn) of both 'a' and
c     'b' to the routine.  Again, this allows us to load array
c     elements in the main program as we have just done,
c     without running into troubles due to the fact that
c     these elements ARE NOT, in general all contiguous in
c     memory.  This certainly includes the current 3 x 3 case.
c-----------------------------------------------------------
      n    = 3
      nrhs = 1

      call dgesv( n, nrhs, a, maxn, ipiv, b, maxn, info )

      if(      info .eq. 0 ) then
c-----------------------------------------------------------
c        Solution successful, write soln to stdout.
c        Note the use of "implied-do-loop" to write a
c        sequence of elements: the enclosing parenthesis
c        around the  "loop" are required.
c-----------------------------------------------------------
         write(*,*) ( b(i) , i = 1 , n )
      else if( info .lt. 0 ) then
c-----------------------------------------------------------
c        Bad argument detected.
c-----------------------------------------------------------
         write(0,*) 'tdgesv1: Argument ', abs(info),
     &              ' to dgesv() is invalid'
      else
c-----------------------------------------------------------
c        Matrix is singular.
c-----------------------------------------------------------
         write(0,*) 'tdgesv1: dgesv() detected singular ',
     &              'matrix'
      end if


      stop

      end
```

Source file: `dgesv.f`

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*  -- LAPACK driver routine (version 2.0) --
*     Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*     Courant Institute, Argonne National Lab, and Rice University
*     March 31, 1993
*
*     .. Scalar Arguments ..
      INTEGER            INFO, LDA, LDB, N, NRHS
*     ..
*     .. Array Arguments ..
      INTEGER            IPIV( * )
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * )
*     ..
*
*  Purpose
*  =======
*
*  DGESV computes the solution to a real system of linear equations
*     A * X = B,
*  where A is an N-by-N matrix and X and B are N-by-NRHS matrices.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as
*     A = P * L * U,
*  where P is a permutation matrix, L is unit lower triangular, and U is
*  upper triangular.  The factored form of A is then used to solve the
*  system of equations A * X = B.
*
*  Arguments
*  =========
*
*  N       (input) INTEGER
*          The number of linear equations, i.e., the order of the
*          matrix A.  N >= 0.
*
*  NRHS    (input) INTEGER
*          The number of right hand sides, i.e., the number of columns
*          of the matrix B.  NRHS >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the N-by-N coefficient matrix A.
*          On exit, the factors L and U from the factorization
*          A = P*L*U; the unit diagonal elements of L are not stored.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  IPIV    (output) INTEGER array, dimension (N)
*          The pivot indices that define the permutation matrix P;
*          row i of the matrix was interchanged with row IPIV(i).
*
*  B       (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
*          On entry, the N-by-NRHS matrix of right hand side matrix B.
*          On exit, if INFO = 0, the N-by-NRHS solution matrix X.
*
*  LDB     (input) INTEGER
*          The leading dimension of the array B.  LDB >= max(1,N).
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = i, U(i,i) is exactly zero.  The factorization
*                has been completed, but the factor U is exactly
*                singular, so the solution could not be computed.
*
*  =====================================================================
*
*     .. External Subroutines ..
      EXTERNAL           DGETRF, DGETRS, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX
*     ..
*     .. Executable Statements ..
*
*     Test the input parameters.
*
      INFO = 0
      IF( N.LT.0 ) THEN
         INFO = -1
      ELSE IF( NRHS.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      ELSE IF( LDB.LT.MAX( 1, N ) ) THEN
         INFO = -7
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGESV ', -INFO )
         RETURN
      END IF
*
*     Compute the LU factorization of A.
*
      CALL DGETRF( N, N, A, LDA, IPIV, INFO )
      IF( INFO.EQ.0 ) THEN
*
*        Solve the system A*X = B, overwriting B with X.
*
         CALL DGETRS( 'No transpose', N, NRHS, A, LDA, IPIV, B, LDB,
     $                INFO )
      END IF
      RETURN
*
*     End of DGESV
*
      END
```

Source file: lnx-output

```
#############################################################
# Building 'tdgesv' and sample output on lnx1
##############################################################
lnx1 1> pwd; ls
/home/phys410/linsys/ex1
Makefile   tdgesv.f

lnx1 2> printenv LIBBLAS
-lblas

lnx1 3> cat Makefile
#############################################################
#  IMPORTANT: Note the use of LIBBLAS which should be
#  set to '-lblas' on the SGI and Linux machines.
#  BLAS is a acronym for Basic Linear Algebra Subprograms
#  and is a Fortran- and C-callable library which implements
#  basic manipulations useful in numerical linear algebra.
#############################################################
.IGNORE:

F77_COMPILE  = $(F77) $(F77FLAGS) $(F77CFLAGS)
F77_LOAD     = $(F77) $(F77FLAGS) $(F77LFLAGS)

.f.o:
$(F77_COMPILE) $*.f

EXECUTABLES = tdgesv

all: $(EXECUTABLES)

tdgesv: tdgesv.o
$(F77_LOAD) tdgesv.o -llapack $(LIBBLAS) -o tdgesv

clean:
rm *.o
rm $(EXECUTABLES)

lnx1 4> make
pgf77 -g -c tdgesv.f
pgf77 -g -L/usr/local/PGI/lib tdgesv.o -llapack -lblas -o tdgesv

lnx1 5> tdgesv
    5.426364412431639      -0.3257753768173936      -0.4083508069894625
```

Source file: sun-output

```
#############################################################
# Building 'tdgesv' and sample output on physics, a Sun 4
# Ultra-Enterprise running SunOS 5.8
#############################################################
physics% pwd; ls
/home2/phys410/linsys/ex1
Makefile   tdgesv.f

physics% make
f77 -O -c tdgesv.f
tdgesv.f:
 MAIN tdgesv1:
f77 -O -L/home/choptuik/lib tdgesv.o -llapack -lblas -o tdgesv

physics% tdgesv
     5.4263644124316  -0.32577537681739  -0.40835080698946
```