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         tdgesv1

      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: `sgi-output`

```
###############################################################
# Building 'tdgesv' and sample output on sgi1, an SGI
# Octane, running IRIX64 6.4
################################################################
sgi1% pwd; ls
/usr/people/phys410/linsys/ex1
Makefile   tdgesv.f

sgi1% printenv LIBBLAS
-lblas

sgi1% 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 Support
#  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)

sgi1% make
f77 -g -64 -c tdgesv.f
f77 -g -64 -L/usr/local/lib tdgesv.o -llapack -lblas -o tdgesv

sgi1% tdgesv
 5.426364412431639  -0.3257753768173935 -0.4083508069894624
```

Source file: `linux-output`

```
###############################################################
# Building 'tdgesv' and sample output on vnfe1, a Trisum
# PIII running Linux 2.2.14.
################################################################
vnfe1% pwd; ls
/home/phys410/linsys/ex1
Makefile  tdgesv.f

vnfe1% printenv LIBBLAS
-lblas

vnfe1% 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 Support
#  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)

vnfe1% make
f77 -g -c tdgesv.f
f77 -g -L/usr/local/lib tdgesv.o -llapack -lblas -o tdgesv

vnfe1% tdgesv
  5.42636441 -0.325775377 -0.408350807
```

Source file: `sun-output`

```
###############################################################
# Building 'tdgesv' and sample output on physics, a Sun 4
# Ultra-Enterprise running SunOS 5.6
################################################################
physics% pwd; ls
/export/ugrad/phys410/linsys/ex1
Makefile   tdgesv.f

physics% make
f77 -O -c tdgesv.f
tdgesv.f:
 MAIN tdgesv1:
f77 -O -L/home5/choptuik/lib tdgesv.o -llapack  -o tdgesv
Undefined first referenced
 symbol       in file
dscal_        /usr/local/lib/liblapack.a(dgetf2.o)
dswap_        /usr/local/lib/liblapack.a(dlaswp.o)
dtrsm_        /usr/local/lib/liblapack.a(dgetrf.o)
idamax_       /usr/local/lib/liblapack.a(dgetf2.o)
dgemm_        /usr/local/lib/liblapack.a(dgetrf.o)
dger_         /usr/local/lib/liblapack.a(dgetf2.o)
ld: fatal: Symbol referencing errors. No output written to tdgesv
make: [tdgesv] Error 1 (ignored)
###############################################################
# OOPS ... those are references to BLAS routines!  I haven't
# defined the environment variable LIBBLAS which is used in the
# Makefile.  Best to add a line in ~/.cshrc.user.  Arguably,
# '-lblas' should just be in the Makefile.  However, on SOME
# systems, BLAS is effectively "built-in", and then explicit
# reference to it can cause problems at load time.  With the
# current mechanism, we can easily deal with such a case
# simply by leaving the environment variable undefined on
# those systems.
###############################################################
physics% vi ~/.cshrc.user

     INSERT ---->   setenv  LIBBLAS  '-lblas'


###############################################################
# "Activate" the changed .cshrc_user; among other things this
# will set LIBBLAS properly.
###############################################################
physics% source !$
source ~/.cshrc.user

[47]physics{phys410} set prompt="physics% "

physics% pwd; ls
/export/ugrad/phys410/linsys/ex1
Makefile   tdgesv.f   tdgesv.o

###############################################################
# This time the build works ...
###############################################################
physics% make
f77 -O -L/home5/choptuik/lib tdgesv.o -llapack -lblas -o tdgesv

###############################################################
# ... and we get output consistent with the other systems.
###############################################################
physics% tdgesv
     5.4263644124316  -0.32577537681739  -0.40835080698946
```

Source file: bvp1d.f

```
c===========================================================
c     Solves 1-d linear boundary value problem
c
c         u''(x) = f(x)  on  x = [0,1]; u(0) = u0, u(1) = u1
c
c     using second-order finite difference technique and
c     LAPACK tridiagonal solver DGTSV.
c===========================================================
      program        bvp1d

      implicit       none

      integer        i4arg

c-----------------------------------------------------------
c     Extrema of problem domain; note that this approach
c     of defining extrema as parameters makes it easier
c     to generalize program to arbitrary domains.
c-----------------------------------------------------------
      real*8         xmin,          xmax
      parameter    ( xmin = 0.0d0,  xmax = 1.0d0 )
c-----------------------------------------------------------
c     Maximum problem size (2**20 + 1)
c-----------------------------------------------------------
      integer        maxn
      parameter    ( maxn = 1 048 577 )
c-----------------------------------------------------------
c     Storage for discrete x-values, unknowns, exact
c     solution and right hand side values
c-----------------------------------------------------------
      real*8         x(maxn),       u(maxn),
     &               uexact(maxn),  f(maxn)
c-----------------------------------------------------------
c     Storage for main, upper and lower diagonals of
c     tridiagonal system, and right-hand-side vector
c     for use with LAPACK routine DGTSV
c-----------------------------------------------------------
      real*8         d(maxn),       du(maxn),
     &               dl(maxn),      rhs(maxn)
      integer        nrhs,          info
c-----------------------------------------------------------
c     Discretization level and size of system (# of discrete
c     unknowns)
c-----------------------------------------------------------
      integer        level,         n,             j,
     &               option
c-----------------------------------------------------------
c     Mesh spacing and related constants (1/h**2, -2/h**2)
c-----------------------------------------------------------
      real*8         h,             hm2,           m2hm2
      real*8         rmserr
c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      level = i4arg(1,-1)
      if( level .lt. 0 ) go to 900
      n = 2 ** level + 1
      if( n .gt. maxn ) then
         write(0,*) 'Insufficient internal storage'
         stop
      end if
      option = i4arg(2,0)
c-----------------------------------------------------------
c     Set up finite-difference 'mesh' (discrete x-values)
c     and define some useful constants.
c-----------------------------------------------------------
      h     =  1.0d0 / (n - 1)
      do j = 1 , n
         x(j) = xmin + (j - 1) * h
      end do
      hm2   =  1.0d0 / (h * h)
      m2hm2 = -2.0d0 / (h * h)
c-----------------------------------------------------------
c     This only ensures that x(n) = xmax EXACTLY and is not
c     essential.
c-----------------------------------------------------------
      x(n) = xmax
c-----------------------------------------------------------
c     Set up exact solution and right hand side vector.
c-----------------------------------------------------------
      call exact(uexact,f,x,n)

c===========================================================
c     Set up tridiagonal system.  Note that indexing on
c     lower diagonal is always (j-1) when implementing the
c     j'th equation.
c===========================================================

c-----------------------------------------------------------
c     Left boundary: u(1) = u_0
c-----------------------------------------------------------
      d(1)       = 1.0d0
      du(1)      = 0.0d0
      rhs(1)     = uexact(1)
c-----------------------------------------------------------
c     Interior: Second order FDA of ODE.
c-----------------------------------------------------------
      do j = 2 , n - 1
         dl(j-1) = hm2
         d(j)    = m2hm2
         du(j)   = hm2
         rhs(j)  = f(j)
      end do
c-----------------------------------------------------------
c     Right boundary: u(n) = u_1
c-----------------------------------------------------------
      dl(n-1)    = 0.0d0
      d(n)       = 1.0d0
      rhs(n)     = uexact(n)

c===========================================================
c     Solve tridiagonal system.
c===========================================================

      nrhs = 1
      call dgtsv( n, nrhs, dl, d, du, rhs, n, info )

      if( info .eq. 0 ) then
c-----------------------------------------------------------
c        Solver successful, output either (x_j, u_j) or
c        (x_j, error_j) to stdout.  Also compute rms error
c        and output to standard error.
c-----------------------------------------------------------
         rmserr = 0.0d0
         do j = 1 , n
            if( option .eq. 0 ) then
               write(*,*) x(j), rhs(j)
            else
               write(*,*) x(j), (uexact(j) - rhs(j))
            end if
            rmserr = rmserr + (uexact(j) - rhs(j)) ** 2
         end do
         rmserr = sqrt(rmserr / n)
         write(0,*) 'rmserr = ', rmserr
      else
c-----------------------------------------------------------
c        Solver failed.
c-----------------------------------------------------------
         write(0,*) 'bvp1d: dgtsv() failed, info = ', info
      end if

      stop

 900  continue
         write(0,*) 'usage: bvp1d <level> [<option>]'
         write(0,*)
         write(0,*) '      Specify option .ne. 0 for output'
         write(0,*) '      of error instead of solution'
      stop

      end
```

```
c===========================================================
c     Computes exact values for u(x) (unknown function)
c     and f(x) (right hand side function).  x array must
c     have been previously defined.
c===========================================================
      subroutine exact(u,f,x,n)

         implicit        none
         integer         n
         real*8          u(n),     f(n),    x(n)

         real*8          pi2
         integer         j

         pi2 = 8.0d0 * atan(1.0d0)
         do j = 1 , n
            u(j) = sin(pi2 * x(j))
            f(j) = -pi2 * pi2 * u(j)
         end do

         return

      end
```

Source file: sgi-output

```
############################################################
# Building 'bvp1d' and sample output on sgi1.
############################################################
sgi1% pwd; ls
/usr/people/phys410/linsys/ex2
Makefile   bvp1d.f    gperr      gpsoln8

sgi1% make
f77 -g -64 -c bvp1d.f
f77 -g -64 -L/usr/local/lib bvp1d.o \
       -lp329f -llapack -lblas -o bvp1d

sgi1% bvp1d
 usage: bvp1d <level> [<option>]

        Specify option .ne. 0 for output
        of error instead of solution

sgi1% bvp1d 4
   0.0000000000000000E+00 -5.5511151231257827E-16
   6.2500000000000000E-02  0.3876394685723090
   0.1250000000000000      0.7162643420150174
   0.1875000000000000      0.9358444623383684
   0.2500000000000000       1.012950746721879
   0.3125000000000000      0.9358444623383684
   0.3750000000000000      0.7162643420150175
   0.4375000000000000      0.3876394685723092
   0.5000000000000000     -2.2204460492503131E-16
   0.5625000000000000     -0.3876394685723097
   0.6250000000000000     -0.7162643420150181
   0.6875000000000000     -0.9358444623383690
   0.7500000000000000      -1.012950746721880
   0.8125000000000000     -0.9358444623383690
   0.8750000000000000     -0.7162643420150181
   0.9375000000000000     -0.3876394685723097
    1.000000000000000     -2.4492935982947064E-16
 rmserr =   8.8841389573651453E-03

############################################################
# Simple convergence test: solve BVP on a sequence of
# levels (h, h/2, h/4, h/16, etc.), redirect stdout to
# /dev/null so that only the overall RMS error appears on
# terminal.  Note how RMS error goes down by very nearly
# a factor of 4 at each successive level, indicating
# O(h^2) convergence.
############################################################
sgi1% foreach level (4 5 6 7 8 9 10)
foreach? bvp1d $level > /dev/null
foreach? end
 rmserr =   8.8841389573651453E-03
 rmserr =   2.2413991373367772E-03
 rmserr =   5.6382739826354859E-04
 rmserr =   1.4145099550532311E-04
 rmserr =   3.5428279660444339E-05
 rmserr =   8.8654982501522291E-06
 rmserr =   2.2174426911240527E-06

############################################################
# Making output files for subsequent plotting via gnuplot.
# See Class Notes for postscript.
############################################################
sgi1% bvp1d 8 > out8
 rmserr =   3.5428279660444339E-05
sgi1% bvp1d 5 1 > err5
 rmserr =   2.2413991373367772E-03
sgi1% bvp1d 6 1 > err6
 rmserr =   5.6382739826354859E-04
sgi1% bvp1d 7 1 > err7
 rmserr =   1.4145099550532311E-04

############################################################
# Gnuplot "script" (gpsoln8) for making plot of level-8
# solution
############################################################
sgi1% cat gpsoln8
set terminal postscript portrait
set size 0.760,1.0
set output "soln8.ps"
plot [0:1] [-1:1] "out8"
quit

############################################################
# Make the plot
############################################################
sgi1% gnuplot < gpsoln8

############################################################
# Gnuplot "script" (gperr) for making plot of error from
# level 5, 6 and 7 calculations
############################################################
sgi1% cat gperr
set terminal postscript portrait
set size 0.760,1.0
set output "err567.ps"
plot "err5", "err6", "err7"
quit

############################################################
# Make the plot
############################################################
sgi1% gnuplot < gperr

sgi1% ls
Makefile    bvp1d.f     err5        err6        gperr       out8
bvp1d*      bvp1d.o     err567.ps   err7        gpsoln8     soln8.ps

############################################################
# Clean-up: Note, the Makefile used here has separate
# 'clean' and 'vclean' (very clean) targets.
############################################################
sgi1% make clean
rm *.o
rm bvp1d

sgi1% ls
Makefile    err5        err6        gperr       out8
bvp1d.f     err567.ps   err7        gpsoln8     soln8.ps

sgi1% make vclean
rm *.o
Cannot access *.o: No such file or directory
make: [clean] Error 2 (ignored)
rm bvp1d
Cannot access bvp1d: No such file or directory
make: [clean] Error 2 (ignored)
rm err[0-9]*
rm out[0-9]*
rm *.ps

sgi1% ls
Makefile   bvp1d.f    gperr      gpsoln8
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = bvp1d

all: $(EXECUTABLES)

bvp1d: bvp1d.o
   $(F77_LOAD) bvp1d.o -lp410f -llapack $(LIBBLAS) -o bvp1d

clean:
   rm *.o
   rm $(EXECUTABLES)

############################################################
# Note the 'vclean' target: 'make vclean' results in
# 'make clean' followed by removal of input and output
# data files and postscript files.
############################################################
vclean: clean
   rm err[0-9]*
   rm out[0-9]*
   rm *.ps
```

Figure file: `../ex2/soln8.ps`

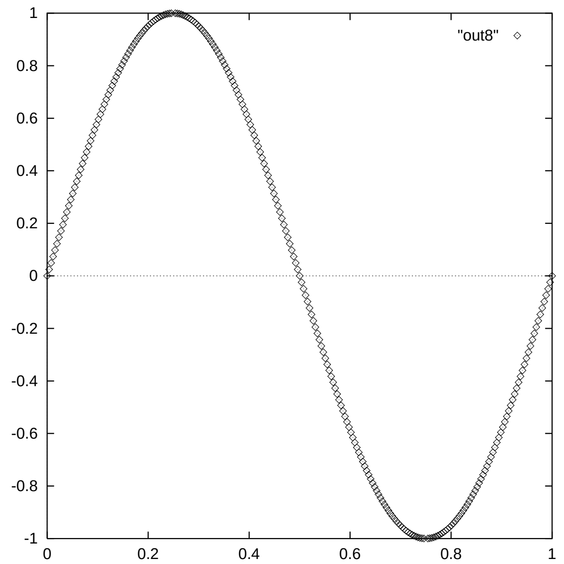
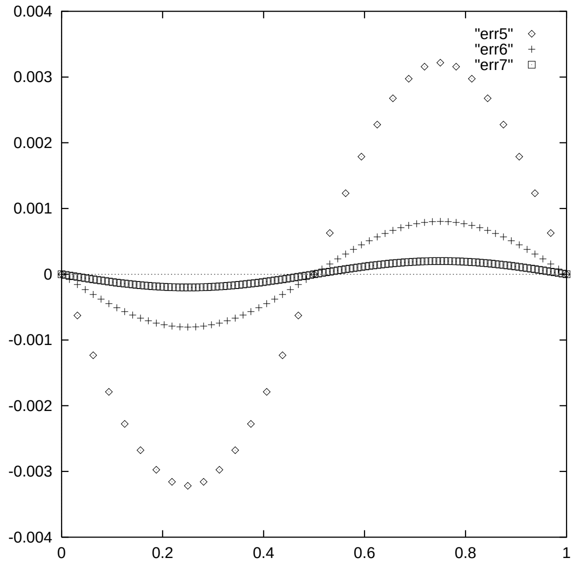


Figure file: `../ex2/err567.ps`



Source file: `dgtsv.f`

```
      SUBROUTINE DGTSV( N, NRHS, DL, D, DU, B, LDB, INFO )
*
*  -- LAPACK routine (version 2.0) --
*     Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*     Courant Institute, Argonne National Lab, and Rice University
*     September 30, 1994
*
*     .. Scalar Arguments ..
      INTEGER            INFO, LDB, N, NRHS
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   B( LDB, * ), D( * ), DL( * ), DU( * )
*     ..
*
*  Purpose
*  =======
*
*  DGTSV  solves the equation
*
*     A*X = B,
*
*  where A is an N-by-N tridiagonal matrix, by Gaussian elimination with
*  partial pivoting.
*
*  Note that the equation  A'*X = B  may be solved by interchanging the
*  order of the arguments DU and DL.
*
*  Arguments
*  =========
*
*  N       (input) INTEGER
*          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.
*
*  DL      (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, DL must contain the (n-1) subdiagonal elements of
*          A.
*          On exit, DL is overwritten by the (n-2) elements of the
*          second superdiagonal of the upper triangular matrix U from
*          the LU factorization of A, in DL(1), ..., DL(n-2).
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, D must contain the diagonal elements of A.
*          On exit, D is overwritten by the n diagonal elements of U.
*
```

```
*  DU      (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, DU must contain the (n-1) superdiagonal elements
*          of A.
*          On exit, DU is overwritten by the (n-1) elements of the first
*          superdiagonal of U.
*
*  B       (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
*          On entry, the N-by-NRHS 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, and the solution
*                has not been computed.  The factorization has not been
*                completed unless i = N.
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ZERO
      PARAMETER          ( ZERO = 0.0D+0 )
*     ..
*     .. Local Scalars ..
      INTEGER            J, K
      DOUBLE PRECISION   MULT, TEMP
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX
*     ..
*     .. External Subroutines ..
      EXTERNAL           XERBLA
*     ..
*     .. Executable Statements ..
*
      INFO = 0
      IF( N.LT.0 ) THEN
         INFO = -1
      ELSE IF( NRHS.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDB.LT.MAX( 1, N ) ) THEN
         INFO = -7
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGTSV ', -INFO )
         RETURN
      END IF
*
      IF( N.EQ.0 )
     $   RETURN
*
      DO 30 K = 1, N - 1
         IF( DL( K ).EQ.ZERO ) THEN
*
*           Subdiagonal is zero, no elimination is required.
*
            IF( D( K ).EQ.ZERO ) THEN
*
*              Diagonal is zero: set INFO = K and return; a unique
*              solution can not be found.
*
               INFO = K
               RETURN
            END IF
         ELSE IF( ABS( D( K ) ).GE.ABS( DL( K ) ) ) THEN
*
*           No row interchange required
*
            MULT = DL( K ) / D( K )
            D( K+1 ) = D( K+1 ) - MULT*DU( K )
            DO 10 J = 1, NRHS
               B( K+1, J ) = B( K+1, J ) - MULT*B( K, J )
   10       CONTINUE
            IF( K.LT.( N-1 ) )
     $         DL( K ) = ZERO
         ELSE
*
*           Interchange rows K and K+1
*
            MULT = D( K ) / DL( K )
            D( K ) = DL( K )
            TEMP = D( K+1 )
            D( K+1 ) = DU( K ) - MULT*TEMP
            IF( K.LT.( N-1 ) ) THEN
               DL( K ) = DU( K+1 )
               DU( K+1 ) = -MULT*DL( K )
            END IF
            DU( K ) = TEMP
            DO 20 J = 1, NRHS
               TEMP = B( K, J )
               B( K, J ) = B( K+1, J )
               B( K+1, J ) = TEMP - MULT*B( K+1, J )
   20       CONTINUE
         END IF
   30 CONTINUE
      IF( D( N ).EQ.ZERO ) THEN
         INFO = N
         RETURN
      END IF
*
*     Back solve with the matrix U from the factorization.
*
      DO 50 J = 1, NRHS
         B( N, J ) = B( N, J ) / D( N )
         IF( N.GT.1 )
     $      B( N-1, J ) = ( B( N-1, J )-DU( N-1 )*B( N, J ) ) / D( N-1 )
         DO 40 K = N - 2, 1, -1
            B( K, J ) = ( B( K, J )-DU( K )*B( K+1, J )-DL( K )*
     $                  B( K+2, J ) ) / D( K )
   40    CONTINUE
   50 CONTINUE
*
      RETURN
*
*     End of DGTSV
*
      END
```

Source file: `bvp1d4.f`

```
c===========================================================
c     Solves 1-d linear boundary value problem
c
c         u''(x) = f(x)  on  x = [0,1]; u(0) = u0, u(1) = u1
c
c     using mixed fourth-order and second order finite
c     difference technique and LAPACK banded solver DGBSV.
c===========================================================
      program         bvp1d4

      implicit        none

      integer         i4arg

c-----------------------------------------------------------
c     Domain extrema and maximum system size.
c-----------------------------------------------------------
      real*8          xmin,          xmax
      parameter     ( xmin = 0.0d0,  xmax = 1.0d0 )

      integer         maxn
      parameter     ( maxn = 2**19 + 1 )
c-----------------------------------------------------------
c     Storage for discrete x-values, unknowns, exact
c     solution and right hand side values.
c-----------------------------------------------------------
      real*8          x(maxn),       u(maxn),
     &                uexact(maxn),  f(maxn)
c-----------------------------------------------------------
c     Number of lower and upper bands.
c-----------------------------------------------------------
      integer         kl,            ku
      parameter     ( kl = 2,        ku = 2  )
c-----------------------------------------------------------
c     Storage for LAPACK-banded-form of linear system,
c     right-hand-side of system and pivot vector,
c     for use with DGBSV.
c
c     Note that for pivoting purposes (row interchanges)
c     DGBSV requires an additional 'kl' rows of workspace.
c     Leading dimension of 'ab' is thus
c
c        ku + kl + kl + 1 = 7
c-----------------------------------------------------------
      integer         ldab
      parameter     ( ldab  = 7 )
      real*8          ab(ldab,maxn), rhs(maxn)
      integer         ipiv(maxn)
c-----------------------------------------------------------
c     Other standard LAPACK parameters.
c-----------------------------------------------------------
      integer         nrhs,          info
c-----------------------------------------------------------
c     Discretization level, size of system (# of discrete
c     unknowns) and output option.
c-----------------------------------------------------------
      integer         level,         n,             option
c-----------------------------------------------------------
c     Storage for difference coefficients.  Note: these
c     arrays have elements -2, -1, 0, 1 and 2.
c-----------------------------------------------------------
      real*8          cdd2(-2:2),    cdd4(-2:2),    c0(-2:2)
c-----------------------------------------------------------
c     Mesh spacing and related constants.
c-----------------------------------------------------------
      real*8          h,             hm2,           hm2by12
c-----------------------------------------------------------
c     Other locals.
c-----------------------------------------------------------
      integer         i,             j,             k
      real*8          rmserr

c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      level = i4arg(1,-1)
      if( level .lt. 0 ) go to 900
      n = 2 ** level + 1
      if( n .gt. maxn ) then
```

```
         write(0,*) 'Insufficient internal storage'
         stop
      end if
      option = i4arg(2,0)

c-----------------------------------------------------------
c     Set up finite-difference 'mesh' (discrete x-values)
c     and difference coefficient arrays.
c-----------------------------------------------------------
      h       =  1.0d0 / (n - 1)
      do j = 1 , n
         x(j) = xmin + (j - 1) * h
      end do
      x(n) = xmax

      hm2     =  1.0d0 / (h * h)
      hm2by12 = hm2 / 12.0d0

      c0(-2)   =  0.0d0
      c0(-1)   =  0.0d0
      c0( 0)   =  1.0d0
      c0( 1)   =  0.0d0
      c0( 2)   =  0.0d0

      cdd2(-2) =  0.0d0
      cdd2(-1) =  hm2
      cdd2( 0) =  -2.0d0 * hm2
      cdd2( 1) =  hm2
      cdd2( 2) =  0.0d0

      cdd4(-2) = -hm2by12
      cdd4(-1) =  16.0d0 * hm2by12
      cdd4( 0) = -30.0d0 * hm2by12
      cdd4( 1) =  16.0d0 * hm2by12
      cdd4( 2) = -hm2by12

c-----------------------------------------------------------
c     Set up exact solution and right hand side vector.
c-----------------------------------------------------------
      call exact(uexact,f,x,n)

c===========================================================
c     Set up banded system.  Recall that for LAPACK
c     banded storage for LU decomposition
c
c     a( i , j ) -> ab( kl + ku + 1 + i - j , j )
c===========================================================

c-----------------------------------------------------------
c     i = 1:  (Left boundary) u(1) = u_0
c-----------------------------------------------------------
      i = 1

      do k = 0 , 2
         j = i + k
         ab(kl + ku + 1 + i - j,j) = c0(k)
      end do
      rhs(i) = uexact(i)
c-----------------------------------------------------------
c     i = 2:  O(h^2) approximation of u''(x) = f(x)
c-----------------------------------------------------------
      i = 2

      do k = -1 , 2
         j = i + k
         ab(kl + ku + 1 + i - j,j) = cdd2(k)
      end do
      rhs(i)  = f(i)
c-----------------------------------------------------------
c     i = 3, ..., n-2: O(h^4) approximation of u''(x) = f(x)
c-----------------------------------------------------------
      do i = 3 , n - 2
         do k = -2 , 2
            j = i + k
            ab(kl + ku + 1 + i - j,j) = cdd4(k)
         end do
         rhs(i) = f(i)
      end do
c-----------------------------------------------------------
c     i = n-1:  O(h^2) approximation of u''(x) = f(x)
c-----------------------------------------------------------
```

```
      i = n - 1

      do k = -2 , 1
         j = i + k
         ab(kl + ku + 1 + i - j,j) = cdd2(k)
      end do
      rhs(i)  = f(i)
c-------------------------------------------------------------
c     i = n:  (Right boundary) u(n) = u_1
c-------------------------------------------------------------
      i = n

      do k = -2 , 0
         j = i + k
         ab(kl + ku + 1 + i - j,j) = c0(k)
      end do
      rhs(i) = uexact(i)

c=============================================================
c     Solve banded system.
c=============================================================

      nrhs = 1
      call dgbsv( n, kl, ku, nrhs, ab, ldab, ipiv, rhs, n,
     &            info )

      if( info .eq. 0 ) then
c-------------------------------------------------------------
c        Solver successful, output either (x_j, u_j) or
c        (x_j, error_j) to stdout.  Also compute rms error
c        and output to standard error.
c-------------------------------------------------------------
         rmserr = 0.0d0
         do j = 1 , n
            if( option .eq. 0 ) then
               write(*,*) x(j), rhs(j)
            else
               write(*,*) x(j), (uexact(j) - rhs(j))
            end if
            rmserr = rmserr + (uexact(j) - rhs(j)) ** 2
         end do
         rmserr = sqrt(rmserr / n)
         write(0,*) 'rmserr = ', rmserr
      else
c-------------------------------------------------------------
c        Solver failed.
c-------------------------------------------------------------
         write(0,*) 'bvp1d4: dgbsv() failed, info = ', info
      end if

      stop

 900  continue
         write(0,*) 'usage: bvp1d4 <level> [<option>]'
         write(0,*)
         write(0,*) '       Specify option .ne. 0 for output'
         write(0,*) '       of error instead of solution'
      stop

      end

c=============================================================
c     Computes exact values for u(x) (unknown function)
c     and f(x) (right hand side function).  x array must
c     have been previously defined.
c=============================================================
      subroutine exact(u,f,x,n)

         implicit        none
         integer         n
         real*8          u(n),     f(n),    x(n)

         real*8          pi2
         integer         j

         pi2 = 8.0d0 * atan(1.0d0)
         do j = 1 , n
            u(j) = sin(pi2 * x(j))
            f(j) = -pi2 * pi2 * u(j)
         end do

         return

      end
```

Source file: sgi-output

```
#############################################################
# Building 'bvp1d4' and sample output on the SGIs
#############################################################
sgi1% pwd; ls
/usr/people/phys410/linsys/ex3
Makefile   bvp1d4.f

sgi1% make
f77 -g -64 -c bvp1d4.f
f77 -g -64 -L/usr/local/lib bvp1d4.o \
    -lp410f -llapack -lblas -o bvp1d4

sgi1% bvp1d4
 usage: bvp1d4 <level> [<option>]

        Specify option .ne. 0 for output
        of error instead of solution

#############################################################
# Note: compare with completely second-order 'bvp1d 4'
# which results in rms error of approximately 9.0E-03.
# These results are about 15 times better at this resolution
# (h = 1/16).
#############################################################
sgi1% bvp1d4 4
   0.0000000000000000E+00 -1.0547118733938987E-14
   6.2500000000000000E-02  0.3834724412118576
   0.1250000000000000      0.7079302872941245
   0.1875000000000000      0.9246563908935262
   0.2500000000000000       1.000689732294703
   0.3125000000000000      0.9244421766816860
   0.3750000000000000      0.7075056502724236
   0.4375000000000000      0.3828904610080090
   0.5000000000000000     -3.1565329029368671E-15
   0.5625000000000000     -0.3828904610080154
   0.6250000000000000     -0.7075056502724303
   0.6875000000000000     -0.9244421766816929
   0.7500000000000000      -1.000689732294711
   0.8125000000000000     -0.9246563908935345
   0.8750000000000000     -0.7079302872941338
   0.9375000000000000     -0.3834724412118676
    1.000000000000000     -2.4492935982947064E-16
 rmserr =   5.8394829778013078E-04

#############################################################
# Convergence test: Solve BVP on a sequence of levels,
# redirect stdout so that only overall RMS error appears
# on terminal.  Rate of convergence is not as definitive
# as it was for the second order calculation, but clearly
# this method converges much more rapidly than the second
# order method.
#############################################################
sgi1% foreach level (4 5 6 7 8 9 10)
foreach? bvp1d4 $level > /dev/null
foreach? end
 rmserr =   5.8394829778013078E-04
 rmserr =   2.5181486533000874E-05
 rmserr =   1.1531108217752065E-06
 rmserr =   5.8557438669720841E-08
 rmserr =   3.2465293440184777E-09
 rmserr =   1.8966747918271887E-10
 rmserr =   9.4215663095393918E-12

#############################################################
# Making output files for subsequent plotting via gnuplot.
# See previous handout for 'bvp1d' for typical 'gnuplot'
# "script" files.
#############################################################
sgi1% bvp1d4 4 > out4
 rmserr =   5.8394829778013078E-04
sgi1% bvp1d4 4 1 > err4
 rmserr =   5.8394829778013078E-04
sgi1% bvp1d4 5 1 > err5
 rmserr =   2.5181486533000874E-05
```

```
sgi1% bvp1d4 6 1 > err6
 rmserr =   1.1531108217752065E-06
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = bvp1d4

all: $(EXECUTABLES)

bvp1d4: bvp1d4.o
   $(F77_LOAD) bvp1d4.o -lp410f -llapack $(LIBBLAS) -o bvp1d4

clean:
   rm *.o
   rm $(EXECUTABLES)

vclean: clean
   rm err[0-9]*
   rm out[0-9]*
   rm *.ps
```

Figure file: `../ex3/soln4.ps`

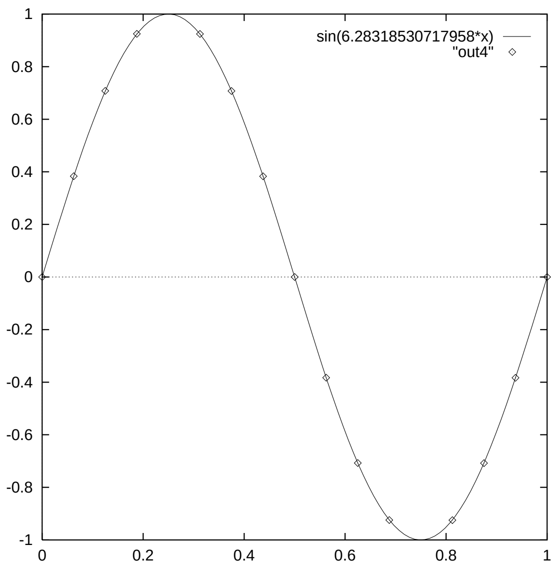


Figure file: `../ex3/err456.ps`