```
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 = 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     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, related constants and locals.
c-----------------------------------------------------------
      real*8          h,            hm2,          hm2by12
      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'
      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)
         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(0) = 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:  (Left boundary) u(1) = 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
```

```
#################################################################
# Building 'bvp1d' and sample output on the SGIs
#################################################################

einstein% pwd; ls
/usr2/people/phy329/linsys/ex3
Makefile   bvp1d4.f

einstein% make
   f77 -g -c bvp1d4.f
   f77 -g -L/usr/local/lib bvp1d4.o \
       -lp329f -llapack -lblas -o bvp1d4

einstein% 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).
#################################################################
einstein%  bvp1d 4
   0.0000000000000000E+00 -2.1094237467877974E-15
   6.2500000000000000E-02  0.3834724412118644
   0.1250000000000000      0.7079302872941298
   0.1875000000000000      0.9246563908935299
   0.2500000000000000       1.000689732294706
   0.3125000000000000      0.9244421766816876
   0.3750000000000000      0.7075056502724246
   0.4375000000000000      0.3828904610080095
   0.5000000000000000     -3.1565329029368671E-15
   0.5625000000000000     -0.3828904610080158
   0.6250000000000000     -0.7075056502724310
   0.6875000000000000     -0.9244421766816937
   0.7500000000000000      -1.000689732294711
   0.8125000000000000     -0.9246563908935347
```

```
   0.8750000000000000     -0.7079302872941337
   0.9375000000000000     -0.3834724412118674
    1.000000000000000      0.0000000000000000E+00
 rmserr =   5.8394829778185022E-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.
#############################################################
einstein% foreach level (4 5 6 7 8 9 10)
foreach? bvp1d4 $level > /dev/null
foreach? end
 rmserr =   5.8394829778185022E-04
 rmserr =   2.5181486528917311E-05
 rmserr =   1.1531108254396499E-06
 rmserr =   5.8557458836146244E-08
 rmserr =   3.2464482687331829E-09
 rmserr =   1.8917403769211032E-10
 rmserr =   9.3084206800586618E-12

#############################################################
# Making output files for subsequent plotting via gnuplot.
# See Class Notes for postscript and previous days notes
# for typical 'gnuplot' script files.
#############################################################
einstein% bvp1d4 4 > out4
 rmserr =   5.8394829778185022E-04

einstein% bvp1d4 4 1 > err4
 rmserr =   5.8394829778185022E-04

einstein% bvp1d4 5 1 > err5
 rmserr =   2.5181486528917311E-05

einstein% bvp1d4 6 1 > err6
 rmserr =   1.1531108254396499E-06
```