```
c===========================================================
c
c     'gs2d' solves the 2-d model boundary-value-problem
c
c       u_xx + u_yy = f(x,y) on [0,1] x [0,1]
c
c       with homogeneous Dirichlet boundary conditions
c
c     using Gauss-Seidel relaxation.  Program has option
c     to either use lexicographic (l) or red-black (c)
c     ordering, as well as various initialization options.
c     See usage message below for more details.
c
c     The following quantities are written to standard
c     output after each iteration (except the first):
c
c     (1) iteration
c     (2) ||error||_2
c     (3) ||running residual||_2
c     (4) rho = estimated spectral radius
c     (5) 1/R = 1/convergence rate = # of iterations to
c         reduce ||error||_2 by 10
c     (6) 1/R * h^2:  Should approach constant as
c         h -> 0 if rho = 1 - O(h^2) as claimed.
c
c===========================================================
      program        gs2d

      implicit       none

      include       'prob.inc'

      integer        iargc,      i4arg
      real*8         r8arg,      dmnrm2

      real*8         relaxc2,    relaxl2

      integer        level,      nsweep,
     &               order,      init
```

```
      integer         default_order
      parameter     ( default_order = 0 )

      integer         default_init
      parameter     ( default_init = 0 )

      logical         ltrace
      parameter     ( ltrace = .true. )

c------------------------------------------------------------
c     Labels for relaxation orderings and initialization
c     options.  Note that intitialization in-place violates
c     the strict f77 standard, but I consider this a
c     safe extension.
c------------------------------------------------------------
      character*32    orderlabel(0:1)
     &               /
     &                'lexicographic',
     &                'red/black'
     &               /
      character*32    initlabel(0:2)
     &               /
     &                'u^(0) = 0',
     &                'u^(0) = uxct',
     &                'u^(0) = random'
     &               /

c------------------------------------------------------------
c     'gft_trace' controls of full output of grid functions
c     using RNPL/BBHUTIL interface for NCSA/NCAR HDF/NETCDF
c     file format.  Can then visualize functions using
c     Iris Explorer and locally developed modules.  More
c     details will be provided via the Course Notes/Software
c     pages.
c------------------------------------------------------------
      logical         gft_trace
      parameter     ( gft_trace = .true. )
```

```fortran
c-----------------------------------------------------------
c     Storage for approximate solution, exact solution,
c     right hand side and true residual.   Note:  All
c     storage defined as one-dimensional; all (2d) array
c     manipulation is to be performed  by subroutines or
c     functions.
c-----------------------------------------------------------
      integer         maxn,               maxsize
      parameter     ( maxn = 2**10 + 1,   maxsize = maxn * maxn)

      real*8          u(maxsize),         uxct(maxsize),
     &                uerr(maxsize),      rhs(maxsize),
     &                res(maxsize)

c-----------------------------------------------------------
c     For storing shape of grid functions.
c-----------------------------------------------------------
      integer         shape(2)

c-----------------------------------------------------------
c     Actual number of grid-points per edge of mesh and
c     mesh spacing.
c-----------------------------------------------------------
      integer         n
      real*8          h

c-----------------------------------------------------------
c     Other locals
c-----------------------------------------------------------
      integer         iter
      real*8          uerrnrm,    rresnrm,  rho,      R
```

```
c-----------------------------------------------------------
c     Argument parsing and echoing.
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

      order   = i4arg(2,default_order)
      if( order .ne. 0  .and.  order .ne. 1 ) goto 900
      nsweep  = i4arg(3,n)
      if( nsweep .le. 0 ) nsweep = n
      init    = i4arg(4,default_init)
      if( init .ne. 0  .and.  init .ne. 1  .and.
     &    init .ne. 2 ) goto 900

      if( ltrace ) then
         write(0,1000) level, n, orderlabel(order), nsweep,
     &                 initlabel(init)
1000     format(' gs2d: Level: ',i2,'  n: ',i4/
     &          ' ge2d: Sweeping order: ',a/
     &          ' ge2d: # of sweeps: ',i5/
     &          ' ge2d: Initialization: ',a
     &         )
      end if

c-----------------------------------------------------------
c     Set mesh spacing, grid function shape
c-----------------------------------------------------------
      h = 1.0d0 / (n - 1)
      shape(1) = n
      shape(2) = n
```

```
c-------------------------------------------------------------
c     Compute exact solution and corresponding right
c     hand side.
c-------------------------------------------------------------
      call clcu2(uxct,h,n,n)
      call clcrhs2(rhs,h,n,n)

c-------------------------------------------------------------
c     Initialize solution estimate and ensure that boundary
c     values are set correctly.
c-------------------------------------------------------------
      if(       init .eq. 0 ) then
         call dmls(u,0.0d0,n,n)
      else if ( init .eq. 1 ) then
         call dmcopy(uxct,u,n,n)
      else if ( init .eq. 2 ) then
         call dmrand(u,n,n)
      else
         write(0,*) 'gs2d: Unexpected init value ', init
         stop
      end if
      call dmlbs(u,0.0d0,n,n)

c-------------------------------------------------------------
c     Dump exact solution, initial guess and
c     right-hand-side.
c-------------------------------------------------------------
      if( gft_trace ) then
         call gft_write('uxct',0.0d0,shape,2,uxct)
         call gft_write('u',0.0d0,shape,2,u)
         call gft_write('rhs',0.0d0,shape,2,rhs)
      end if
```

```
c-------------------------------------------------------------
c     R E L A X A T I O N   L O O P
c-------------------------------------------------------------
      do iter = 1 , nsweep
c-------------------------------------------------------------
c        Perform one relaxation sweep of selected type.
c-------------------------------------------------------------
         if(      order .eq. 0 ) then
            rresnrm = relaxl2(u,rhs,n,n,h)
         else if( order .eq. 1 ) then
            rresnrm = relaxc2(u,rhs,n,n,h)
         else
            write(0,*) 'gs2d: Unexpected sweep order ',
     &                order
            stop
         end if

c-------------------------------------------------------------
c        Compute solution error and norm of error.
c-------------------------------------------------------------
         call dmms(uxct,u,uerr,n,n)
         uerrnrm = dmnrm2(uerr,n,n)

c-------------------------------------------------------------
c        Compute convergence rate.
c-------------------------------------------------------------
         call clcrhor(uerrnrm,rho,R)

c-------------------------------------------------------------
c        Tracing output ... for convenience, don't count
c        the first iteration.
c-------------------------------------------------------------
            if( iter .eq. 1 ) then
               write(0,2000)
2000           format(
     & ' Iter    |err|      |res|         rho           '
     & '1/R      h^2/R'
     &              )
            else
               write(*,2100) iter - 1, uerrnrm,  rresnrm, rho,
```

```
     &                       1.0d0 / r,
     &                       1.0d0 / (r * (n - 1)**2)
2100           format(i4,1p,2e11.2,e16.6,e11.2,e13.4,0p)
            end if
c-------------------------------------------------------------
c        Dump estimate of solution and error.
c-------------------------------------------------------------
         if( gft_trace ) then
            call gft_write('u',1.0d0 * iter,shape,2,u)
            call gft_write('uerr',1.0d0 * iter,shape,2,u)
         end if

      end do
c-------------------------------------------------------------
c     Normal exit
c
c     Need to call this routine to ensure that .hdf files
c     are closed properly.
c-------------------------------------------------------------
      call gft_close_all()
      stop

c-------------------------------------------------------------
c     Usage exit
c-------------------------------------------------------------
 900  continue
         write(0,*) 'usage: gs2d <level> [<order> <nsweep> '//
     &              '<init>]'
         write(0,*)
         write(0,*) '       <order> = 0  lexical order (default)'
         write(0,*) '       <order> = 1  red-black order'
         write(0,*)
         write(0,*) '       <nsweep> defaults to n = '//
     &              '# of grid pts. per edge of mesh'
         write(0,*)
         write(0,*) '       <init> = 0    u^(0) = 0 (default)'
         write(0,*) '       <init> = 1    u^(0) = uxct'
         write(0,*) '       <init> = 2    u^(0) = random'
      stop
      end
```

```
c-----------------------------------------------------------
c     Problem-specific parameters for 'gs2d'
c-----------------------------------------------------------
      real*8            pi
      parameter       ( pi = 3.141 5926 5358 9793 d0 )

      real*8            lx,           ly
      parameter       ( lx = 1.0d0,    ly = 2.0d0 )
```

```
c-----------------------------------------------------------
c     Performs lexicographic GS relaxation sweep of 2d model
c     problem and returns norm of running ('instantaneous')
c     residuals.
c-----------------------------------------------------------
      double precision function relaxl2(u,rhs,nx,ny,h)

         implicit       none

         integer        nx,              ny
         real*8         u(nx,ny),        rhs(nx,ny)
         real*8         h

         real*8         hm2,      mjelm1,   rresl
         integer        i,        j

         relaxl2 = 0.0d0
         hm2     = 1.0d0 / (h * h)
         mjelm1   =  h * h / 4.0d0

         do j = 2 , ny - 1
            do i = 2 , nx - 1
               rresl = hm2 * (u(i+1,j) + u(i-1,j) +
     *                       u(i,j+1) + u(i,j-1) -
     *                       4.0d0 * u(i,j)) -
     *                rhs(i,j)
               u(i,j) = u(i,j) + mjelm1 *  rresl
               relaxl2 = relaxl2 + rresl * rresl
            end do
         end do
         relaxl2 = sqrt(relaxl2 / (nx * ny))

         return

      end
```

```
c-----------------------------------------------------------
c     Performs red-black GS relaxation sweep of 2d model
c     problem and returns norm of running ('instantaneous')
c     residuals.
c-----------------------------------------------------------
      double precision function relaxc2(u,rhs,nx,ny,h)
         implicit       none

         integer        nx,              ny
         real*8         u(nx,ny),        rhs(nx,ny)
         real*8         h

         real*8         hm2,      mjelm1,   rresl
         integer        i,        j,
     *                  isw,      jsw,      pass

         relaxc2 = 0.0d0
         hm2     = 1.0d0 / (h * h)
         mjelm1   =  h * h / 4.0d0

         jsw = 1
          do pass = 1 , 2
            isw = jsw
            do j = 2 , ny - 1
               do i = isw + 1 , nx - 1, 2
                  rresl = hm2 * (u(i+1,j) + u(i-1,j) +
     *                           u(i,j+1) + u(i,j-1) -
     *                           4.0d0 * u(i,j)) -
     *                    rhs(i,j)
                  u(i,j) = u(i,j) + mjelm1 *  rresl
                  relaxc2 = relaxc2 + rresl * rresl
               end do
               isw = 3 - isw
            end do
            jsw = 3 - jsw
         end do
         relaxc2 = sqrt(relaxc2 / (nx * ny))

         return
      end
```

```
c-----------------------------------------------------------
c     Calculates exact solution:
c
c        u(x,y) = sin(l_x pi x) sin(l_y pi y)
c-----------------------------------------------------------
      subroutine clcu2(u,h,nx,ny)

         implicit       none

         include       'prob.inc'

         integer        nx,       ny
         real*8         u(nx,ny)

         real*8         h

         integer        i,        j

         do  j = 1 , ny
            do  i = 1 , nx
                  u(i,j) = sin(pi * lx * (i - 1) * h) *
     *                     sin(pi * ly * (j - 1) * h)
            end do
         end do
         call dmlbs(u,0.0d0,nx,ny)

         return

      end
```

```
c-------------------------------------------------------------
c     Calculates right hand side corresponding to above
c     exact solution.
c-------------------------------------------------------------
      subroutine clcrhs2(rhs,h,nx,ny)

         implicit       none

         include       'prob.inc'

         integer        nx,       ny
         real*8         rhs(nx,ny)

         real*8         h

         real*8         uij
         integer        i,        j

         do  j = 1 , ny
            do  i = 1 , nx
               uij =
     *                    sin(pi * lx * (i - 1) * h) *
     *                    sin(pi * ly * (j - 1) * h)
               rhs(i,j) = -(lx * lx + ly * ly)
     *                          * pi * pi * uij
            end do
         end do
         call dmlbs(rhs,0.0d0,nx,ny)

         return

      end
```

```
c-----------------------------------------------------------
c     Loads "border" of 2--array with scalar.
c-----------------------------------------------------------
      subroutine dmlbs(a,s,d1,d2)

         implicit      none

         integer       d1,         d2
         real*8        a(d1,d2)

         real*8        s

         integer       i1,         i2

         do i2 = 1 , d2
            a(1, i2) = s
            a(d1,i2) = s
         end do

         do i1 = 1 , d1
            a(i1,1)  = s
            a(i1,d2) = s
         end do

         return

      end
```

```
c-------------------------------------------------------------
c     Load matrix with scalar.
c-------------------------------------------------------------
      subroutine dmls(a,s,d1,d2)

         implicit       none

         integer        d1,       d2
         real*8         a(d1,d2)
         real*8         s

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a(i,j) = s
            end do
         end do

         return

      end
```

```fortran
c-----------------------------------------------------------------
c     Load matrix with random values between -1.0d0
c     and 1.0d0. SGI-specific.
c-----------------------------------------------------------------
      subroutine dmrand(a,d1,d2)

         implicit       none

         real*8         rand

         integer        d1,       d2
         real*8         a(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a(i,j) = 2.0d0 * rand() - 1.0d0
            end do
         end do

         return

      end
```

```
c-------------------------------------------------------------
c     Matrix-matrix subtract.
c-------------------------------------------------------------
      subroutine dmms(a1,a2,a3,d1,d2)

         implicit       none

         integer        d1,         d2
         real*8         a1(d1,d2),  a2(d1,d2), a3(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a3(i,j) = a1(i,j) - a2(i,j)
            end do
         end do

         return

      end
```

```fortran
c-----------------------------------------------------------
c     Matrix-matrix copy.
c-----------------------------------------------------------
      subroutine dmcopy(a1,a2,d1,d2)

         implicit       none

         integer        d1,        d2
         real*8         a1(d1,d2), a2(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a2(i,j) = a1(i,j)
            end do
         end do

         return

      end
```

```
c-------------------------------------------------------------
c     Matrix l2-norm.
c-------------------------------------------------------------
      double precision function dmnrm2(a,d1,d2)

         implicit       none

         integer        d1,       d2
         real*8         a(d1,d2)

         integer        i,        j

         dmnrm2 = 0.0d0

         do j = 1 , d2
            do i = 1 , d1
               dmnrm2 = dmnrm2 + a(i,j) * a(i,j)
            end do
         end do
         dmnrm2 = sqrt(dmnrm2 / (d1 * d2))

         return

      end
```

```fortran
c-----------------------------------------------------------
c     Computes estimate of spectral radius rho(G) of
c     residual amplification matrix and convergence rate
c     R = log10(1/rho) whose reciprocal estimates number of
c     iterations required to take norm(residuals) down by an
c     order of magnitude.  Returns (0.0, 0.0) on
c     initial call.
c-----------------------------------------------------------
      subroutine clcrhor(resnrm,rho,r)
         implicit     none
         real*8       resnrm, rho, r

c-----------------------------------------------------------
c        Maintains previous residual norm.
c-----------------------------------------------------------
         real*8       presnrm

         logical      first / .true. /
         save         presnrm,    first

         if( first ) then
            first = .false.
            rho = 0.0d0
            r   = 0.0d0
         else
            if( resnrm .ne. 0.0d0 ) then
               rho = resnrm / presnrm
               r   = log10(1.0d0 / rho)
            else
               rho = 0.0d0
               r   = 0.0d0
            end if
         end if
         presnrm = resnrm

         return
      end
```