```fortran
c=============================================================
c     sode:  Sample driver for integrating ODEs using
c     ODEPACK routine LSODA.
c
c     LSODA solves the initial value problem for systems
c     of first order ODEs:
c
c          dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(neq))
c
c             i = 1, 2, ... neq
c
c     The user is responsible for providing a subroutine
c     called 'fcn' which computes the right-hand-sides
c     of the ODEs as discussed in the class notes.  If
c     the order of the system is not 2, then the parameter
c     'neq' below must be changed appropriately.
c
c     This program expects initial values to be specified
c     on the command-line, optionally followed by an
c     error tolerance.  It reads a vector of times
c     at which the solution is to be output from standard
c     input, one number per line.  Output to standard
c     output is
c
c         t,   y_1,   y_2   ...   y_neq
c
c     for each requested output time 't' (neq + 1 numbers
c     per line).
c=============================================================
      program        sode

      implicit       none
```

```
c-----------------------------------------------------------
c     Codename: Change this parameter string in accord with
c     the program name.  Use a fixed-length character
c     string of the appropriate length.
c-----------------------------------------------------------
      character*4    cdnm
      parameter    ( cdnm = 'sode' )

      integer        iargc,         indlnb,         i4arg
      real*8         r8arg

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )
c-----------------------------------------------------------
c     Order of system.
c-----------------------------------------------------------
      integer        neq
      parameter    ( neq = 2 )
c-----------------------------------------------------------
c     Storage for solution at requested output times.
c-----------------------------------------------------------
      integer        maxout
      parameter    ( maxout = 10 000 )
      real*8         y0(neq)
      real*8         vtout(maxout), vyout(maxout,neq),
     &               work(maxout)
      integer        ntout,         itout
      integer        ieq

      logical        ltrace
      parameter    ( ltrace = .false. )

      integer        maxdump
      parameter    ( maxdump = 50 )

      logical        lsodatrace
      parameter    ( lsodatrace = .false. )

      logical        ivs_ok
```

```
c-----------------------------------------------------------
c     LSODA Variables.
c-----------------------------------------------------------
      external        fcn,        jac

      real*8          y(neq)
      real*8          tbgn,       tend
      integer         itol
      real*8          rtol,       atol
      integer         itask,      istate,     iopt
      integer         lrw

      parameter     ( lrw = 22 + neq * 16 )
      real*8          rwork(lrw)

      integer         liw
      parameter     ( liw = 20 + neq )
      integer         iwork(liw)
      integer         jt

      real*8          tol
      real*8          default_tol
      parameter     ( default_tol = 1.0d-6 )

c-----------------------------------------------------------
c     Common communication with routine 'fcn' in 'fcn.f' ...
c-----------------------------------------------------------
      include        'fcn.inc'

                       .
                       .
                       .
```

```
c===========================================================
c     Some double precision vector utility routines.
c
c     Originally from ~matt/vutil/dveclib.f
c===========================================================

c-----------------------------------------------------------
c     Dumps vector labelled with LABEL on UNIT.
c-----------------------------------------------------------
      subroutine dvdump(v,n,label,unit)
         implicit       none

         real*8         v(*)
         character*(*)  label
         integer        i, n, st, unit

         if( n .lt. 1 ) go to 130
            write(unit,100) label
 100        format(/' <<< ',A,' >>>'/)
            st = 1
 110        continue
               write(unit,120) ( v(i) , i = st , min(st+3,n))
 120           FORMAT(' ',4(1PE19.10))
               st = st + 4
            if( st .le. n ) go to 110

 130     continue

         return
      end
```

```
c-----------------------------------------------------------
c     Extension of dvdump which dumps every 'inc'th element.
c-----------------------------------------------------------
      subroutine dvdmp1(v,w,inc,n,label,unit)
         implicit       none

         real*8         v(*),    w(*)
         character*(*)  label
         integer        inc,     n,      unit

         call dvinj(v,w,inc,n)
         call dvdump(w,1+(n-1)/inc,label,unit)

         return

      end

c---------------------------------------------------------
c     Injects every inc'th element of v1 into v2
c---------------------------------------------------------
      subroutine dvinj(v1,v2,inc,n)
         implicit    none

         real*8      v1(*),     v2(*)
         integer     i,         inc,     j,      n

         j = 1
         do i = 1 , n , inc
            v2(j) = v1(i)
            j = j + 1
         end do

         return

      end
```

```
c===========================================================
c     orbit:  Integrates restricted gravitational 2-body
c     problem.  See 'fcn.f' for more details on the equations
c     of motion.   Initial values (specified on command line)
c     are
c
c          x(0)  v_x(0)  y(0)  v_y(0)
c
c     Output to standard out is
c
c          t   x(t)  v_x(t)  y(t)  v_y(t)
c
c     for all requested output times 't' read from standard
c     input.
c===========================================================
      program        orbit

      implicit       none
c-----------------------------------------------------------
c     Codename: Change this parameter string in accord with
c     the program name.  Use a fixed-length character
c     string of the appropriate length.
c-----------------------------------------------------------
      character*5    cdnm
      parameter    ( cdnm = 'orbit' )

      integer        iargc,         indlnb,         i4arg
      real*8         r8arg

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )
c-----------------------------------------------------------
c     Order of system.
c-----------------------------------------------------------
      integer        neq
      parameter    ( neq = 4 )
```

.
.
.

```
c---------------------------------------------------------------
c     Common communication with routine 'fcn' in 'fcn.f' ...
c---------------------------------------------------------------
      include       'fcn.inc'

c--------------------------------------------------------------------------
c     Initialize parameters defined  in common block ...
c--------------------------------------------------------------------------
      G = 1.0d0
      M = 1.0d0
```

```
c-----------------------------------------------------------------------
c     Application specific common block for communication with
c     derivative evaluating routine 'fcn' (optional) ...
c-----------------------------------------------------------------------

      real*8  G,          M
      common / com_fcn /
     &        G,          M
```

```fortran
c-----------------------------------------------------------------------
c     Implements (planar) equations of motion for restricted 2-body
c     gravitational problem.  Central mass, M,  fixed at (0,0).
c     Mass of other object with coordinates (x_c,y_c) is gravitationally
c     negligible. ' denotes differentiation with respect to t.
c
c     y(1) := x_c
c     y(2) := x_c'
c     y(3) := y_c
c     y(4) := y_c'
c-----------------------------------------------------------------------
      subroutine fcn(neq,t,y,yprime)
         implicit    none

c-----------------------------------------------------------------------
c        Problem parameters (G, M) passed in via common block defined
c        in 'fcn.inc'
c-----------------------------------------------------------------------
         include    'fcn.inc'

         integer     neq
         real*8      t,     y(neq),    yprime(neq)

         real*8      c1

         c1 = -G * M / (y(1)**2 + y(3)**2)**1.5d0

         yprime(1) = y(2)
         yprime(2) = c1 * y(1)
         yprime(3) = y(4)
         yprime(4) = c1 * y(3)

         return
      end
```

```
c----------------------------------------------------------------------
c     Application specific common block for communication with
c     derivative evaluating routine 'fcn' (optional) ...
c----------------------------------------------------------------------

      real*8
     &         MM,    m1bym2,   d,     G,
     &         d1,    d2,       mu,
     &         m1,    m2,
     &         x1,    x2,       y1,    y2
      common / com_fcn /
     &         MM,    m1bym2,   d,     G,
     &         d1,    d2,       mu,
     &         m1,    m2,
     &         x1,    x2,       y1,    y2
```

```
c-----------------------------------------------------------------------
c     Solves EOM for orbiting dumbbell (rigid body composed of
c     2 point masses m1 and m2, separation d)
c
c     See class notes for equations of motion.
c
c-----------------------------------------------------------------------
      subroutine fcn(neq,t,y,yprime)
         implicit    none

         include    'fcn.inc'

         integer     neq
         real*8      t,     y(neq),    yprime(neq)

         real*8      xc,    yc,    th,
     &               c1,    c2,
     &               r1m3,  r2m3

         xc = y(1)
         yc = y(3)
         th = y(5)

         x1 = xc + d1 * cos(th)
         y1 = yc + d1 * sin(th)
         x2 = xc - d2 * cos(th)
         y2 = yc - d2 * sin(th)

         r1m3 = 1.0d0 / (x1**2 + y1**2) ** 1.5d0
         r2m3 = 1.0d0 / (x2**2 + y2**2) ** 1.5d0
```

```
    c1 = -G * MM
    c2 =  G * MM / d

    yprime(1) = y(2)
    yprime(2) = c1 * ((1.0d0 - mu) * x1 * r1m3 +
&                              mu  * x2 * r2m3)
    yprime(3) = y(4)
    yprime(4) = c1 * ((1.0d0 - mu) * y1 * r1m3 +
&                              mu  * y2 * r2m3)
    yprime(5) = y(6)
    yprime(6) = c2 * (r1m3 - r2m3) *
&                    (sin(th) * xc - cos(th) * yc)

    xc = y(1)
    yc = y(3)
    th = y(5)

    x1 = xc + d1 * cos(th)
    y1 = yc + d1 * sin(th)
    x2 = xc - d2 * cos(th)
    y2 = yc - d2 * sin(th)

    return
 end
```