Source file: `dumb.f`

```
c===========================================================
c     Integrates equations of motion for orbiting dumbbell.
c
c     Performs output via 'vsxynt' interface.  Currently
c     configured to write to .sedgat files which can
c     then be sent to 'ser' or 'jser' visualization servers.
c===========================================================
      program        dumb

      implicit       none

      character*2    itoc
      character*64   catsqz
      integer        iargc,         indlnb,         vsxynt
      integer        vsrc

      real*8         x_vs(3),       y_vs(3)

      integer        stderr,      stdin,     stdout
      parameter    ( stderr = 0,  stdin = 5, stdout = 6 )

      real*8         s2r8
      real*8         r8arg
      integer        s2i4
      integer        i4arg
      character*128  sarg

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )

      integer        i4_never
      parameter    ( i4_never = -2 000 000 000 )

      integer       neq
      parameter   ( neq = 6 )

      integer       maxout
      parameter   ( maxout = 50 000 )

      real*8        y0(neq)
      real*8        vtout(maxout), vyout(maxout,neq),
     &              work(maxout),  ketrans(maxout),
     &              kerot(maxout), pegrav(maxout),
     &              etot(maxout)
      integer       ntout,         itout

      integer       ieq

      logical       ltrace
      parameter   ( ltrace = .true. )

      integer       maxdump
      parameter   ( maxdump = 50 )

      logical       vstrace
      parameter   ( vstrace = .true. )

      logical       lsodatrace
      parameter   ( lsodatrace = .false. )

      logical       ivs_ok

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

      real*8        y(neq),     yprime(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     Hard-code some of the parameters
c-----------------------------------------------------------
      G  = 1.0d0
      MM = 1.0d0

c-----------------------------------------------------------
c     Parse command line arguments (initial values) followed
c     by m1/m2, d, tol
c-----------------------------------------------------------
      if( iargc() .lt. neq ) then
         write(0,*) 'dumb: Insufficient initial values ',
     &              'specified on command line.'
         go to 900
      end if

      ivs_ok = .true.
      do ieq = 1 , neq
         y0(ieq) = r8arg(ieq,r8_never)
         if( y0(ieq) .eq. r8_never ) then
            write(0,*) 'dumb: Initial value ', ieq,
     &                 ' unrecognizable as a real number.'
            ivs_ok = .false.
         end if
      end do

      if( .not. ivs_ok ) go to 900

      m1bym2 = r8arg(neq+1,0.5d0)
      mu     = 1.0d0 / (1.0d0 + m1bym2)
      d      = r8arg(neq+2,0.1d0)
      tol    = r8arg(neq+3,default_tol)

      m1     = 1.0d0
      m2     = m1bym2

      d1     = 1.0d0 / (1.0d0 + m1bym2) * d
      d2     = d - d1

      if( ltrace ) then
         write(0,2000) G, MM, m1bym2, d, d1, d2, tol
2000     format('Gravitational constant:  ',F8.2/
     &          'Central mass: ',F8.2/
     &          'm1/m2: ',F12.5 /
     &          'Dumbbell separation: ',F12.5/
     &          'd1: ',F12.5/
     &          'd2: ',F12.5/
     &          'LSODA tolerance: ',1P,E10.2)
         call dvdump(y0,neq,'dumb: initial values',0)
      end if

c-----------------------------------------------------------
c     Get output times from standard input ...
c-----------------------------------------------------------
      call dvfrom('-',vtout,ntout,maxout)
      if( ntout .le. 0 ) then
         write(0,*)
         write(0,*) 'dumb: No output times read from stdin'
         write(0,*) 'dumb: Use (e.g.) dvmesh or dvgmesh to '//
     &              'generate output times.'
         write(0,*)
         write(0,*) 'dumb: Sample usages:'
         write(0,*)
         write(0,*) 'dumb: dvmesh  0.0 1.0 101 | dumb ... '
         write(0,*) 'dumb: dvgmesh 0.0 1.0 101 10 | dumb ...'
         write(0,*)
         stop
      end if
      if( ltrace ) then
```

```
         if( ntout .le. maxdump ) then
            call dvdump(vtout,ntout,'dumb: output times',0)
         else
            call dvdmp1(vtout,work,
     &                  int(1.0d0 * ntout / maxdump + 0.5d0),
     &                  ntout,'dumb: selected output times',0)
         end if
         write(0,*)
         write(0,*) '***** INITIAL TIME: ', vtout(1)
         write(0,*)
      end if

c---------------------------------------------------------------
c     Set LSODA parameters ...
c---------------------------------------------------------------
      itol = 1
      rtol = tol
      atol = tol
      itask = 1
      iopt = 0
      jt = 2

c---------------------------------------------------------------
c     Initialize the solution ...
c---------------------------------------------------------------
      call dvcopy(y0,y,neq)

c---------------------------------------------------------------
c     ... and call the external routine to initialize the
c     auxiliary quantities.
c---------------------------------------------------------------
      call fcn(neq,vtout(1),y,yprime)

c---------------------------------------------------------------
c     Do the integration ...
c---------------------------------------------------------------
      do itout = 1 , ntout
         do ieq = 1 , neq
            vyout(itout,ieq) = y(ieq)
         end do
c---------------------------------------------------------------
c        Compute the total energy ... normalized for m2 = 1,
c        m1 = m2bym1.
c---------------------------------------------------------------
         ketrans(itout) = 0.5d0 * (m1 + m2) *
     &                    (y(2)**2 + y(4)**2)
         kerot(itout)   = 0.5d0 * (m1 * m2) / (m1 + m2) *
     &                    (d * y(6))**2
         pegrav(itout)  = - G * MM *
     &                    (m1 / sqrt(x1**2 + y1**2) +
     &                     m2 / sqrt(x2**2 + y2**2))
         etot(itout) = ketrans(itout) + kerot(itout) +
     &                 pegrav(itout)
         if( itout .eq. ntout ) go to 500

         istate = 1
c---------------------------------------------------------------
c        Need these temporaries since lsoda overwrites
c        tend ...
c---------------------------------------------------------------
         tbgn = vtout(itout)
         tend = vtout(itout+1)
         call lsoda(fcn,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
         if( lsodatrace ) then
            write(0,1000) itout, ntout, vtout(itout), vtout(itout+1)
1000        format(' dumb: Step ',i4,'  t = ',1pe10.3,' .. ',1pe10.3)
            write(0,*) 'dumb: lsoda reurns ', istate
         end if

         if( istate .lt. 0 ) then
            write(0,1500) istate, itout, ntout, vtout(itout),
     &                    vtout(itout+1)
1500        format(/' dumb: Error return ',i2,' from integrator LSODA.'/
     &              ' dumb: At output time ',i5,' of ',i5/
     &              ' dumb: Interval ',1pe11.3,' .. ',1pe11.3/)
            go to 500
         end if
         if( vstrace ) then
            x_vs(1) = x1
            x_vs(2) = x2

            y_vs(1) = y1
            y_vs(2) = y2
            vsrc = vsxynt('dumb',tend,x_vs,y_vs,2)
         end if

      end do

500   continue
      if( vstrace ) then
         vsrc = vsxynt('th',0.0d0,vtout,vyout(1,5),itout)
         vsrc = vsxynt('om',0.0d0,vtout,vyout(1,6),itout)
         vsrc = vsxynt('etot',0.0d0,vtout,etot,itout)
         vsrc = vsxynt('ketrans',0.0d0,vtout,ketrans,itout)
         vsrc = vsxynt('kerot',0.0d0,vtout,kerot,itout)
         vsrc = vsxynt('pegrav',0.0d0,vtout,pegrav,itout)
      end if

      write(*,*) vtout(itout), ( vyout(itout,ieq) , ieq = 1 , neq )
      stop

900   continue
         write(0,*) 'usage: dumb <x_0> <vx_0> <y_0> <vy_0> '//
     &              ' <th_0> <vth_0> [<m1/m2> <d> tol]'
      stop
      end
```

Source file: `fcn.f`

```
c=========================================================
c     Solves EOM for orbiting dumbbell (rigid body composed
c     of 2 point masses m1 and m2, separation d)
c
c     See class notes for equations of motion.
c
c     Canonicalization:
c
c        y(1) = xc
c        y(2) = d(xc)/dt
c        y(3) = yc
c        y(4) = d(yx)/dt
c        y(5) = th
c        y(6) = d(th)/dt
c=========================================================
c     neq = 6
c---------------------------------------------------------

c---------------------------------------------------------
c     Implements differential equations.
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

c===========================================================
c     Implements Jacobian  (optional) ...
c===========================================================
      subroutine jac
         implicit    none

         include    'fcn.inc'

         return
      end
```

Source file: `fcn.inc`

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

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = dumb

all: $(EXECUTABLES)

LIBS = -lfvs -lutilio -lvutil -lodepack -llinpack -lblas
dumb: dumb.o fcn.o fcn.inc
   $(F77_LOAD) dumb.o fcn.o $(LIBS) -o dumb

clean:
   cleantex
   /bin/rm dumb
   /bin/rm *.o
   /bin/rm *.segdat
```

Figure file: `om_c.ps`

Angular frequency, ω, of dumbbell about its center of mass, versus time, t, for a circular orbit.

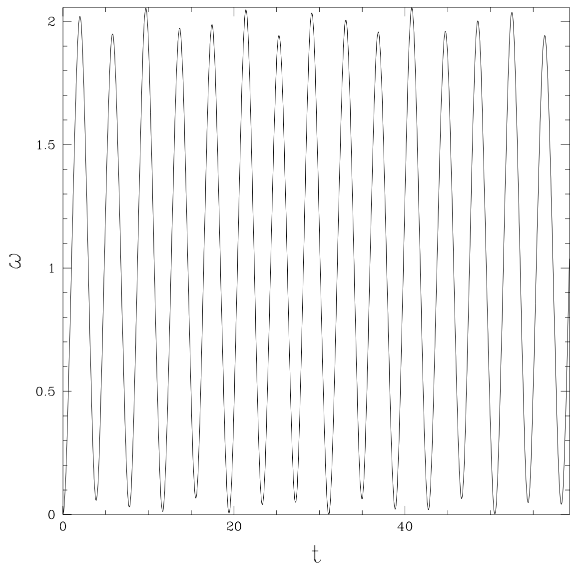

Figure file: `om_e.ps`

Angular frequency, ω, of dumbbell about its center of mass, versus time, t, for an elliptical orbit. Note the "chaotic" nature of ω in this case.

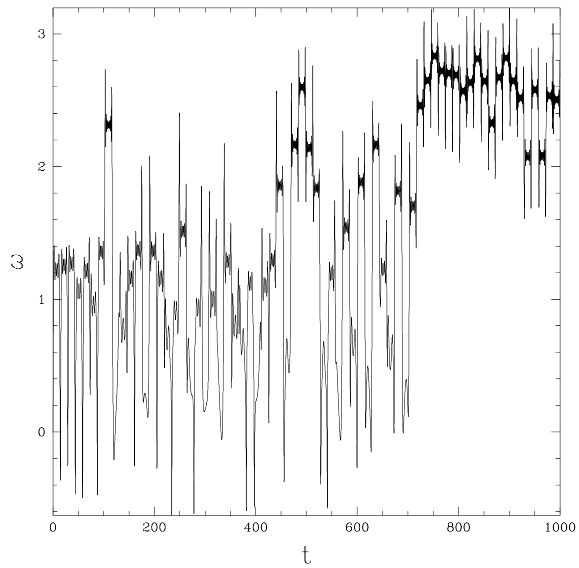

Figure file: `om_ez.ps`

Detail of ω for the elliptical orbit

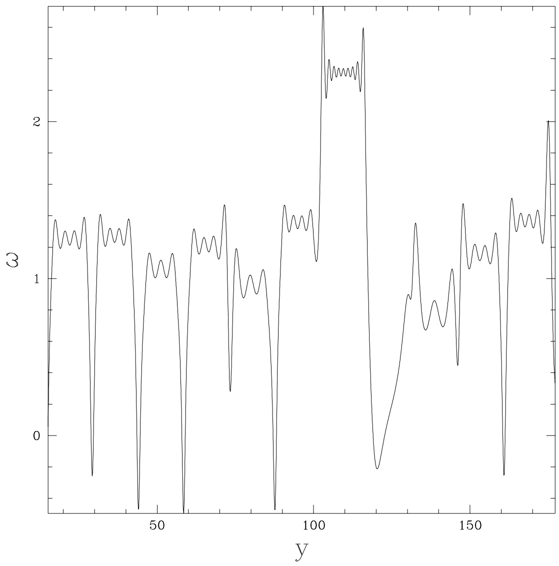

Figure file: `e_6.ps`

Energy quantities computed with `LSODA` *tolerance* `1.0e-6`. *Top to bottom: Translational kinetic energy,* $T_{\rm trans}$*, rotational kinetic energy,* $T_{\rm rot}$*, total energy,* $E_{\rm tot}$*, and gravitational potential energy,* $V_{\rm grav}$*. In this case, "non-conservation" of total mechanical energy is a good sign that we need to make the* `LSODA` *error tolerance more stringent.*

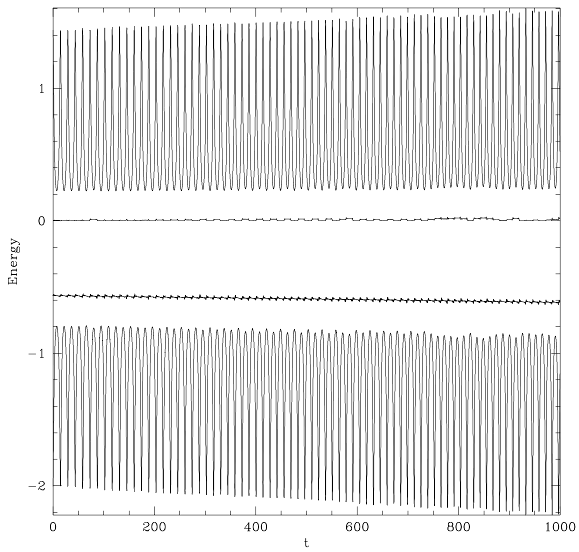

Figure file: `e_12.ps`

Energy quantities computed with `LSODA` *tolerance* `1.0e-12`. *Note that energy conservation is much improved in this case.*

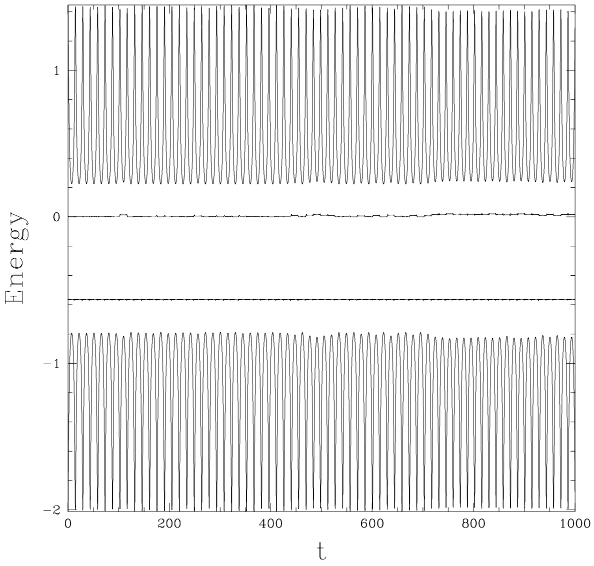

Figure file: `kerot_12.ps`

Plot of rotational kinetic energy, $T_{\rm rot}$*, computed with* `LSODA` *tolerance* `1.0e-12`. *Again, note the "chaotic" appearance of the energy.*