Source file: `twobody.f`

```
c===========================================================
c     twobody:  Integrates restricted gravitational 2-body
c     problem using LSODA.
c
c     usage: twobody <x0> <y0> <vx0> <vy0> <tmax> <dt> [<tol>]
c
c     Output to standard output
c
c        0.0    x(0.0)   y(0.0)   dEtot(0.0)  dJtot(0.0)
c        dt     x(dt)    y(dt)    dEtot(dt)   dJtot(dt)
c       2*dt    x(2*dt)  y(2*dt)  dEtot(2*dt) dJtot(2*dt)
c                          .
c                          .
c                          .
c       tmax    x(tmax)  y(tmax)  dEtot(tmax) dJtot(tmax)
c===========================================================
      program       twobody

      implicit      none

      integer       iargc,      i4arg
      real*8        r8arg

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

c-----------------------------------------------------------
c     Command line arguments (initial position and velocity
c     components will be read directly into y() array).
c-----------------------------------------------------------
      real*8        tmax,       dt,        tol

c-----------------------------------------------------------
c     LSODA Variables.
c-----------------------------------------------------------
      integer       neq
      parameter    ( neq = 4)

      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        default_tol
      parameter    ( default_tol = 1.0d-6 )

c-----------------------------------------------------------
c     Locals
c
c     Etot:   Instantaneous total mechanical energy
c     Jtot:   Instantaneous total angular momentum
c     Etot0:  Initial total mechanical energy
c     Jtot0:  Initial total angular momentum
c-----------------------------------------------------------
      real*8        t,          ts,         tf
      integer       ieq
      real*8        Etot,       Jtot,
     &              Etot0,      Jtot0

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     Parse command line arguments (initial values) ...
c-----------------------------------------------------------
      if( iargc() .lt. 6 ) go to 900

      do ieq = 1 , 4
         y(ieq) = r8arg(ieq,r8_never)
         if( y(ieq) .eq. r8_never ) go to 900
      end do
      tmax = r8arg(5,r8_never)
      if( tmax .eq. r8_never ) go to 900
      dt   = r8arg(6,r8_never)
      if( dt   .eq. r8_never ) go to 900
      tol  = r8arg(7,default_tol)

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

c-----------------------------------------------------------
c     Compute initial energy, angular momentum, then output
c     initial time, particle coordinates,
c     Etot - Etot0 and Jtot - Jtot0
c
c     Note use of format statement to ensure that all five
c     numbers are output on a single line, 'write(*,*)'
c     will break lines, inhibiting further processing with
c     Unix utilities.  The format statement is good for
c     up to 10 numbers per line.
c-----------------------------------------------------------
      t = 0.0d0
      call calc_ej(y,Etot0,Jtot0)
      call calc_ej(y,Etot,Jtot)

      write(*,1000) t, y(1), y(2),
     &              Etot - Etot0, Jtot - Jtot0
1000  format(1P, 10 E25.16, 0P)

c-----------------------------------------------------------
c     Do the integration ...
c-----------------------------------------------------------
      do while( t .le. tmax )
         istate = 1
         ts = t
         tf = t + dt
c-----------------------------------------------------------
c        Integrate EOM from t=ts to t=tf ...
c-----------------------------------------------------------
         call lsoda(fcn,neq,y,ts,tf,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)

c-----------------------------------------------------------
c        Check return code; bail-out with an error message
c        if routine was not successful ...
c-----------------------------------------------------------
         if( istate .lt. 0 ) then
            write(0,*) 'twobody:  Error return ', istate,
     &                 ' from LSODA '
            write(0,*) 'twobody: Current interval ', t, t + dt
            stop
         end if

         t = t + dt
c-----------------------------------------------------------
c        Compute new energy and angular momentum, output
c        as previously (i.e. use the same format statement)
c-----------------------------------------------------------
         call calc_ej(y,Etot,Jtot)
         write(*,1000) t, y(1), y(2),
     &                 Etot - Etot0, Jtot - Jtot0
      end do
```

```
      stop

900   continue
         write(0,*) 'usage: twobody <x0> <y0> <vx0> <vy0> '//
     &              '<tmax> <dt> [<tol>]'
      stop
      end
```

Source file: `fcn.f`

```
c===========================================================
c     Implements (planar) equations of motion for restricted
c     2-body gravitational problem.  Central mass, M, is
c     fixed at (0,0).  Mass of other object with coordinates
c     (x_c,y_c) is gravitationally negligible.
c     ' denotes differentiation with respect to t.
c
c     y(1) := x_c
c     y(2) := y_c
c     y(3) := x_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
c        block defined 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(2)**2)**1.5d0

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

         return
      end

c===========================================================
c     Computes mechanical energy (etot) and angular momentum
c     about the origin (location of the gravitating mass)
c     from the dynamical variables.  "Specific" quantities
c     (i.e. normalized by the mass of the dynamical test
c     particle) are computed.
c===========================================================
      subroutine calc_ej(y,etot,jtot)
         implicit      none
         real*8        y(4),    etot,    jtot

         include      'fcn.inc'

         etot = 0.5d0 * (y(3)**2 + y(4)**2) -
     &          G * M / sqrt(y(1)**2 + y(2)**2)
         jtot = y(1) * y(4) - y(2) * y(3)

         return
      end

c===========================================================
c     Dummy Jacobian routine.
c===========================================================
      subroutine jac
         implicit    none

         include    'fcn.inc'

         return
      end
```

Source file: `fcn.inc`

```
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
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = twobody

all: $(EXECUTABLES)

twobody.o: twobody.f fcn.inc
fcn.o:     fcn.f     fcn.inc

twobody: twobody.o fcn.o fcn.inc
   $(F77_LOAD) twobody.o fcn.o -lp410f -lodepack \
               -llinpack $(LIBBLAS) -o twobody

clean:
   /bin/rm $(EXECUTABLES)
   /bin/rm *.o

vclean: clean
   /bin/rm out_*
   /bin/rm *.ps
```

Source file: `Twobody`

```
#! /bin/sh

############################################################
# This shell script is a "front-end" to twobody which
# expedites the analysis of the results from that code,
# including the generation of Postscript plots of the
# particle position, d(energy), d(angular momentum) as a
# function of time using gnuplot.
############################################################
P=`basename $0`

############################################################
# Set defaults
############################################################
tmax=5.0
dt=0.05
tol=1.0d-6

############################################################
# Usage
############################################################
Usage() {
cat<<END
usage: $P <y0> [<tol>]

       Default tol: $tol

       y0 = 1.0 will produce circular orbit.

       To enable automatic previewing of Postscript files
       set GV environment variable to any non-blank
       value, e.g.

       setenv GV on
END
exit 1
}

############################################################
```

```
# Subroutine (fcn) to produce postscript version of
# gnuplot plot of data stored in file $1.  Postscript
# file will be called $1.ps.  If optional second argument
# is supplied, the resulting Postscript file will be
# 'gv'ed.
############################################################
gnuplot_it() {
gnuplot<<END
   set terminal postscript portrait
   set size square
   set xlabel "x"
   set ylabel "$1"
   set output "$1.ps"
   plot "$1"
   quit
END
if test "${2}undefined" != undefined; then
   if [ -f $1.ps ]; then
      (gv $1.ps) &
   else
       echo "gnuplot_it: $f.ps does not exist"
   fi
fi
}

############################################################
# Argument handling
############################################################
case $# in
1|2) y0=$1; tol=${2-$tol};;
*) Usage;;
esac

############################################################
# Build application, run it, and process the results.
############################################################
make -f Makefile twobody

tag="$y0"-"$tol"
ofile=out-$tag

twobody 0.0 $y0 1.0 0.0 $tmax $dt $tol > $ofile

nth 2 3 < $ofile > xcyc-$tag
nth 1 2 < $ofile > xc-$tag
nth 1 3 < $ofile > yc-$tag
nth 1 4 < $ofile > dEtot-$tag
nth 1 5 < $ofile > dJtot-$tag

for f in xcyc-$tag xc-$tag yc-$tag dEtot-$tag dJtot-$tag; do
   gnuplot_it $f $GV
   /bin/rm $f
done
/bin/ls -l *$tag*.ps

exit 0
```

Figure file: ../twobody/xcyc-1.0-1.0d-6.ps

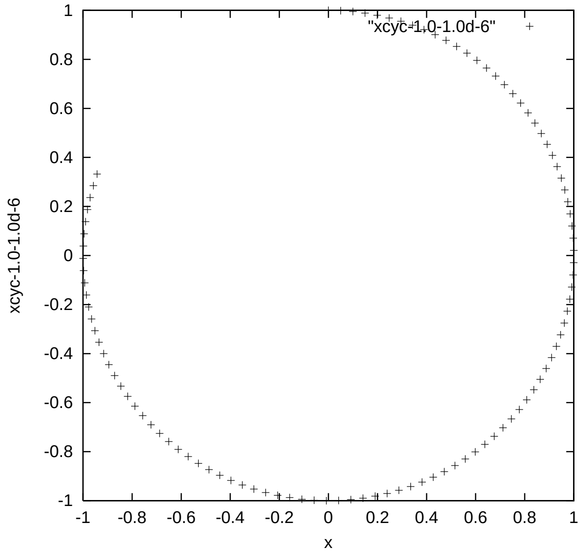


Figure file: ../twobody/dEtot-1.0-1.0d-6.ps

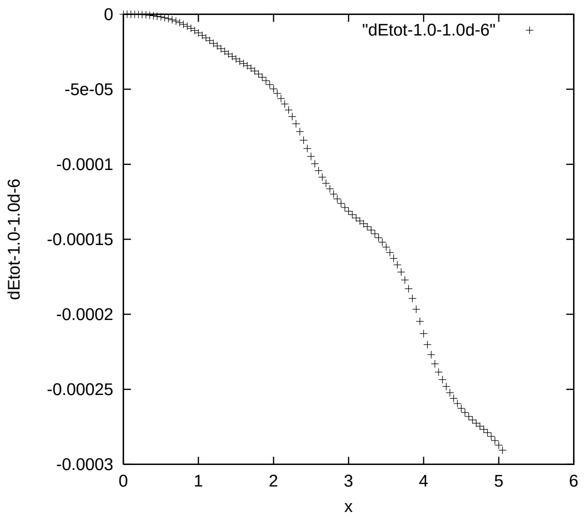


Figure file: `../twobody/dJtot-1.0-1.0d-6.ps`

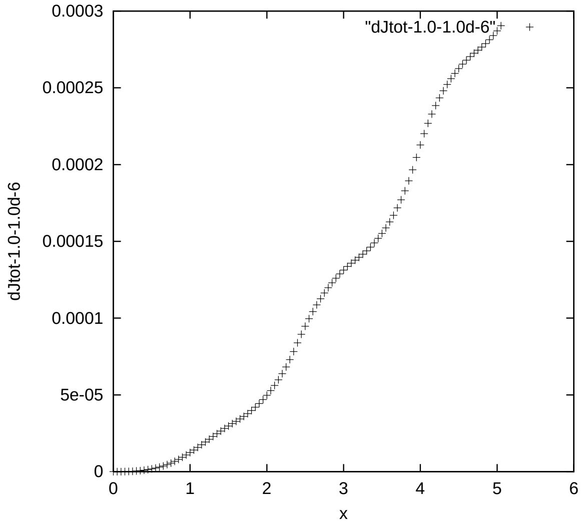


Figure file: `../twobody/dEtot-1.0-1.0d-10.ps`

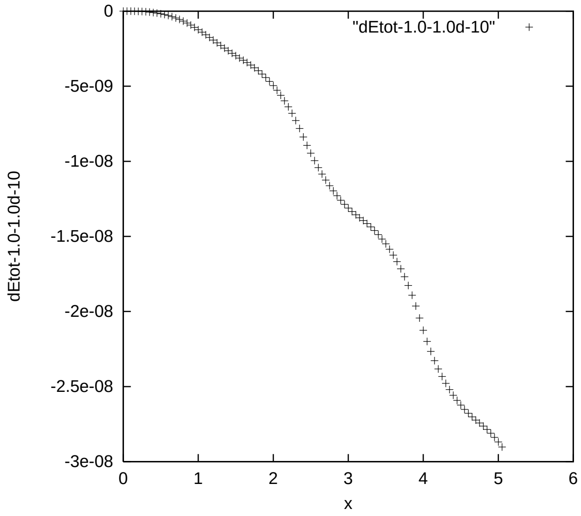


Figure file: `../twobody/xcyc-1.0-1.0d-10.ps`

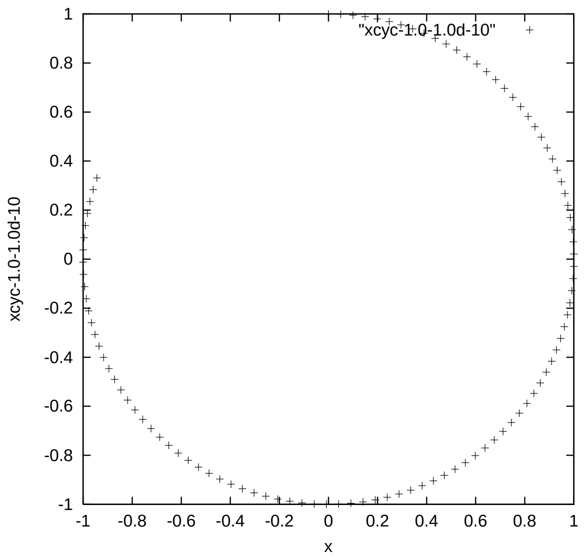


Figure file: `../twobody/dJtot-1.0-1.0d-10.ps`

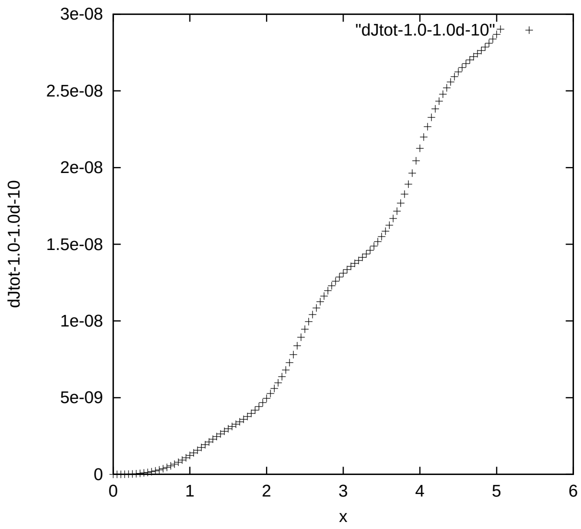


Figure file: `../twobody/xcyc-0.8-1.0d-10.ps`

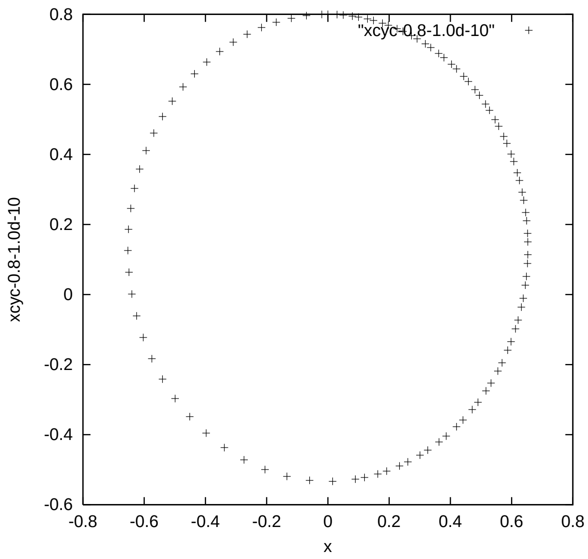


Figure file: `../twobody/dJtot-0.8-1.0d-10.ps`

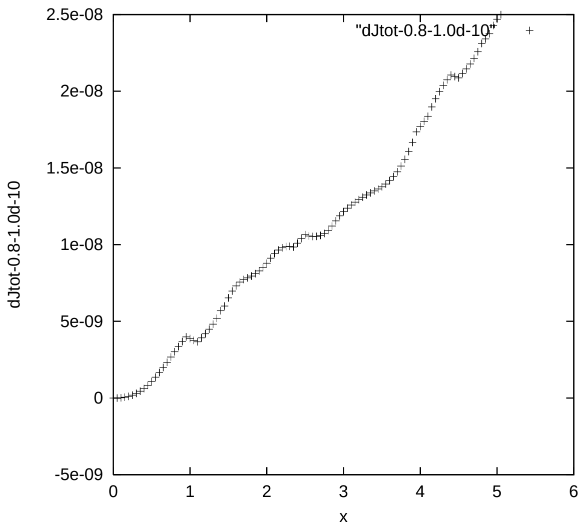
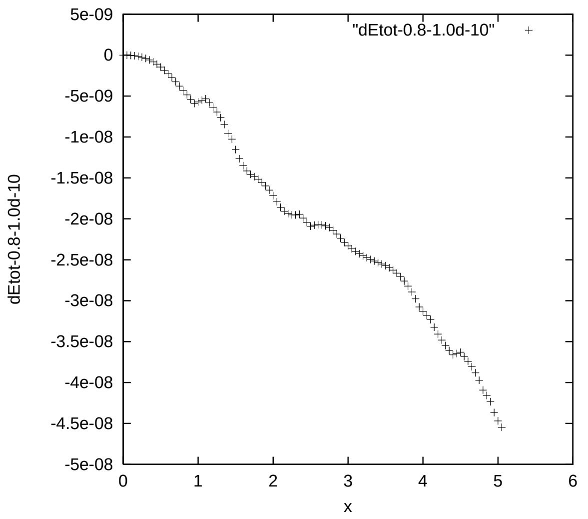


Figure file: `../twobody/dEtot-0.8-1.0d-10.ps`



Source file: dumb.f

```
c===========================================================
c     dumb: Uses LSODA to integrate equations of motion for
c     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
c     Output to standard output is
c
c Column:    1  2  3  4  5 6     7     8    9    10   11
c Quantity:  t x1 y1 x2 y2 theta omega KE_t KE_r PE_g E_tot
c
c     at t=0, dtout, 2 dtout, ... , tmax
c===========================================================
      program        dumb

      implicit       none

      character*2    itoc
      real*8         r8arg
      integer        iargc,         indlnb

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

c-----------------------------------------------------------
c     Command-line arguments
c-----------------------------------------------------------
      real*8         tmax,       dtout

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

      integer        neq
      parameter    ( neq = 6 )

      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     Locals
c-----------------------------------------------------------
      real*8         t,          tout

c-----------------------------------------------------------
c     Parse command line arguments.
c-----------------------------------------------------------
      if( iargc() .ne. 4 ) go to 900

      y(4)  = r8arg(1,r8_never)
      tmax  = r8arg(2,r8_never)
      dtout = r8arg(3,r8_never)
      tol   = r8arg(4,r8_never)
      if( y(4)  .eq. r8_never  .or. tmax .eq. r8_never  .or.
     &    dtout .eq. r8_never  .or. tol  .eq. r8_never )
     &  go to 900
c-----------------------------------------------------------
c     Hard-code the remainder of the problem parameters:
c
c     ( x_c(0),  y_c(0)  )   =  ( 1.0 , 0.0 )
c     ( vx_c(0), vy_c(0) )   =  ( 0.0 , vy0 )
c
c     theta(0) = 0
c     omega(0) = 0
c
c     m1/m2    = 0.5
c     d        = 0.1
c-----------------------------------------------------------
      G  = 1.0d0
      MM = 1.0d0

      y(1)  = 1.0d0
      y(2)  = 0.0d0
      y(3)  = 0.0d0

      y(5)  = 0.0d0
      y(6)  = 0.0d0

c     m1bym2 = 0.5d0
c     Energy not conserved for m1bym2 .ne. 1
      m1bym2 = 2.0d0
      mu     = 1.0d0 / (1.0d0 + m1bym2)
      d      = 0.1d0

      m1     = 1.0d0
      m2     = m1 / m1bym2

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

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

c-----------------------------------------------------------
c     Call the RHS-evaluating routine to initialize the
c     auxiliary quantities, and output initial values.
c-----------------------------------------------------------
      t = 0.0d0
      call fcn(neq,t,y,yprime)
      write(*,1100) t, x1, y1, x2, y2, th, om,
     &              ketrans, kerot, pegrav, etot
1100  format(1P,12E24.16,0P)

c-----------------------------------------------------------
c     Do the integration.
c-----------------------------------------------------------
      do while( t .le. tmax )
         istate = 1
         tout   = t + dtout
         call lsoda(fcn,neq,y,t,tout,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
         if( istate .lt. 0 ) then
            write(0,*) 'dumb:  Error return ', istate,
     &                 ' from LSODA '
            write(0,*) 'dumb: Current interval ',
     &                 t, t + dtout
            stop
         end if
c-----------------------------------------------------------
c     Call the RHS-evaluating routine to compute the
c     auxiliary quantities, and output them.
c-----------------------------------------------------------
         call fcn(neq,t,y,yprime)
         write(*,1100) t, x1, y1, x2, y2, th, om,
     &                 ketrans, kerot, pegrav, etot
      end do

      stop
```

```
900   continue
         write(0,*) 'usage: dumb <y_0> <tmax> <dtout> <tol>'
         write(0,*) ' '
         write(0,*)
     &      '       Specify <y_0> = 1.0 for circular orbit'
      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===========================================================
      subroutine fcn(neq,t,y,yprime)
         implicit    none

         include    'fcn.inc'

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

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

c-----------------------------------------------------------
c        Define some auxiliary quantities to make
c        computation of RHSs more transparent.
c-----------------------------------------------------------
         xc = y(1)
         yc = y(3)
         th = y(5)
         om = y(6)

         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)

c-----------------------------------------------------------
c        Compute positions of two components of the
c        dumbbell.
c-----------------------------------------------------------
         x1 = xc + d1 * cos(th)
         y1 = yc + d1 * sin(th)
         x2 = xc - d2 * cos(th)
         y2 = yc - d2 * sin(th)

c-----------------------------------------------------------
c        Compute the total energy ...
c-----------------------------------------------------------
         ketrans = 0.5d0 * (m1 + m2) *
     &                     (y(2)**2 + y(4)**2)
         kerot   = 0.5d0 * (m1 * m2) / (m1 + m2) *
     &                     (d * y(6))**2
         pegrav  = - G * MM *
     &                     (m1 / sqrt(x1**2 + y1**2) +
     &                      m2 / sqrt(x2**2 + y2**2))
         etot    = ketrans + kerot + pegrav

         return
      end

c===========================================================
c     Dummy Jacobian routine.
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,
     &         th,    om,
     &         ketrans,         kerot,
     &         pegrav,          etot

      common / com_fcn /
     &         MM,    m1bym2,   d,     G,
     &         d1,    d2,       mu,
     &         m1,    m2,
     &         x1,    x2,       y1,    y2,
     &         th,    om,
     &         ketrans,         kerot,
     &         pegrav,          etot
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = dumb

all: $(EXECUTABLES)

dumb: dumb.o fcn.o fcn.inc
   $(F77_LOAD) dumb.o fcn.o -lp410f -lodepack \
            -llinpack -lblas -o dumb

clean:
   /bin/rm dumb
   /bin/rm *.o
   rm *ps
   rm circular
   rm elliptical
   rm elliptical-lo
   rm *_e
   rm *_el
```

Source file: Dumb

```
\documentclass [11pt] {article}
\usepackage{verbatim}
\usepackage{epsfig}
%
%
\topmargin=-0.75in
\hoffset=-0.75in
\textheight=9.0in \vsize\textheight
\textwidth=6.5in \hsize\textwidth
\baselineskip=50pt
\def\lb{\left(}
\def\rb{\right)}
\def\lbb{\left[}
\def\rbb{\right]}
\renewcommand{\baselinestretch}{1.5}
\parindent=0in
%
\begin {document}
{\LARGE
\sf
\newpage
\setcounter{page}{1}
\setcounter{equation}{0}
%
\begin{center}
{\bf PHYS 410/555 Computational Physics:\\ The Orbiting Dumbbell}\\
{(Following Giordano, {\em Computational Physics}, Section 4.6)}
\end  {center}
%

\vspace*{1.0in}
{\em Background:} With the exception of Hyperion, which is one
of Saturn's satellites, all of the moons in the solar system
are ``spin-locked''; a moon which is spin-locked has a rotational
frequency, $\omega$ about its own spin axes which is the same
as its orbital frequency, $\Omega$.  The supposed mechanism
by which the spin-locking comes about is somewhat involved; however,
the point is that Hyperion is somehow exceptional---study of its
$\omega(t)$ suggests that it is tumbling {\em chaotically} in
its orbit about Saturn, which is presumably due to both to its
peculiar shape (like that of an egg), and the fact that it
is in an elliptical orbit about Saturn.

\newpage
To investigate the effects of a non-spherical distribution
of mass on a satellite's spin as it orbits its parental
body, we consider the model of an ``orbiting dumbbell''.

\vspace*{0.2in}
\centerline{\epsfxsize=8cm\epsffile{dumb.eps}}
\vspace*{0.2in}

Consider two test masses, $m_1$, $m_2$ connected by a massless rigid rod
of length $d$, in orbit about a mass, $M >> m_1, m_2$ as shown in the
figure above.
Let $(x_i,y_i), i = 1 , 2$ be the coordinates of the two test masses,
let $(x_c,y_c)$ be the coordinates of the dumbbell's center of mass,
and let $\theta$ be the angle the rod makes with the $x$-axis.
Defining
$$
\mu \equiv \frac{m_2}{m_1 + m_2}
$$
the distances of the masses from the center of mass are
$$
d_1 = \mu \, d \quad\quad\quad d_2 = (1 - \mu) d
$$
then
$$
x_i  =  x_c \pm d_i \cos\theta \quad\quad\quad
y_i  =  y_c \pm d_i \sin\theta
$$
%
The moment of inertia of the dumbbell about $(x_c,y_c)$ is
$$
I = m_1 d_1{}^2 + m_2 d_2{}^2 =
  \frac{m_1 m_2{}^2}{(m_1 + m_2)^2} d{}^2 +
  \frac{m_2 m_1{}^2}{(m_1 + m_2)^2} d{}^2 =
  \frac{m_1 m_2}{m_1 + m_2} d^2
$$
```

```
%
The equations of motion for the body are
$$
\lb m_1 + m_2 \rb {\bf a_c} = \lb m_1 + m_2 \rb {\ddot {\bf r}}_c =
\sum {\bf F} = {\bf F}_1 + {\bf F}_2
$$
$$
I \alpha = I {\ddot \theta} = \sum \tau =
  {\bf d}_1 \times {\bf F}_1 + {\bf d}_2 \times {\bf F}_2
$$
where
$$
{\bf F}_1 = -\frac{G M m_1}{r_1{}^3} \lbb x_1 , y_1 \rbb
$$
$$
{\bf F}_2 = -\frac{G M m_2}{r_2{}^3} \lbb x_2 , y_2 \rbb
$$
are the gravitational forces acting on $m_1$ and $m_2$ respectively.
The translational equations yield:
$$
\lb m_1 + m_2 \rb \, {\ddot x}_c =
   -G M \lb \frac{m_1}{r_1{}^3}x_1 + \frac{m_2}{r_2{}^3} y_2 \rb
$$
$$
\lb m_1 + m_2 \rb \, {\ddot y}_c =
   -G M \lb \frac{m_1}{r_1{}^3}y_1 + \frac{m_2}{r_2{}^3} y_2 \rb
$$
or
$$
{\ddot x}_c = -G M \lb \frac{1 - \mu}{r_1{}^3} x_1 + \frac{\mu}{r_2{}^3} x_2 \rb
$$
$$
{\ddot y}_c = -G M \lb \frac{1 - \mu}{r_1{}^3} y_1 + \frac{\mu}{r_2{}^3} y_2 \rb
$$
The rotational equation gives:
\begin{eqnarray*}
I {\ddot \theta} = {\bf d}_1 \times {\bf F}_1 + {\bf d}_2 \times {\bf F}_2
&=& -G M \frac{m_1}{r_1{}^3} d_1 \lb \cos\theta y_1 - \sin\theta x_1 \rb  + \\
&~& \quad G M \frac{m_2}{r_2{}^3} d_2  \lb \cos\theta y_2 - \sin\theta x_2 \rb \\
&=& -G M \frac{m_1}{r_1{}^3} d_1 \lb \cos\theta y_c - \sin\theta x_c \rb + \\
&~& \quad G M \frac{m_2}{r_2{}^3} d_2 \lb \cos\theta y_c - \sin\theta x_c \rb \\[0
&=& G M \lb \frac{m_2}{r_2{}^3}d_2 - \frac{m_1}{r_1{}^3}d_1 \rb
    \lb \cos\theta y_c - \sin\theta x_c \rb \\[0.2in]
&=& G M \frac{m_1 m_2}{m_1 + m_2} d \lb \frac{1}{r_1{}^3} - \frac{1}{r_2{}^3}\rb
    \lb \sin\theta x_c - \cos\theta y_c \rb
\end{eqnarray*}
so
$$
{\ddot \theta} = \frac{GM}{d} \lb \frac{1}{r_1{}^3} - \frac{1}{r_2{}^3}\rb
    \lb \sin\theta x_c - \cos\theta y_c \rb
$$
\newpage
Summarizing, we have:
\begin{eqnarray*}
{\ddot x}_c &=&
-G M \lb \frac{1 - \mu}{r_1{}^3} x_1 + \frac{\mu}{r_2{}^3} x_2 \rb \\
{\ddot y}_c &=&
-G M \lb \frac{1 - \mu}{r_1{}^3} y_1 + \frac{\mu}{r_2{}^3} y_2 \rb \\
{\ddot \theta} &=& \frac{GM}{d} \lb \frac{1}{r_1{}^3} - \frac{1}{r_2{}^3}\rb
    \lb \sin\theta x_c - \cos\theta y_c \rb
\end{eqnarray*}
where
\begin{eqnarray*}
\mu &\equiv& \frac{m_2}{m_1 + m_2} \\
d_1 &=& \mu \, d \\
d_2 &=& (1 - \mu) d \\
x_i &=& x_c \pm d_i \cos\theta \\
y_i &=& y_c \pm d_i \sin\theta \\
r_i{}^3 &=& \lb x_i{}^2 + y_i{}^2 \rb^{3/2}
\end{eqnarray*}
The total (conserved) energy of the system is
$$
E_{\rm tot} = T_{\rm trans} + T_{\rm rot} + V_{\rm grav}
$$
where
\begin{eqnarray*}
T_{\rm trans} &\equiv&
   \frac{1}{2}\lb m_1 + m_2 \rb \lb {\dot x_c}^2 + {\dot y_c}^2 \rb\\
T_{\rm rot} &\equiv&
   \frac{1}{2}\frac{m_1 m_2}{m_1 + m_2} \, d^2 \, {\dot \theta}^2 \\
V_{\rm grav} &\equiv& -G M \lb \frac{m_1}{r_1} + \frac{m_2}{r_2} \rb
```

```
\end{eqnarray*}
}
\end {document}
```

Figure file: `../dumb/om-c.ps`

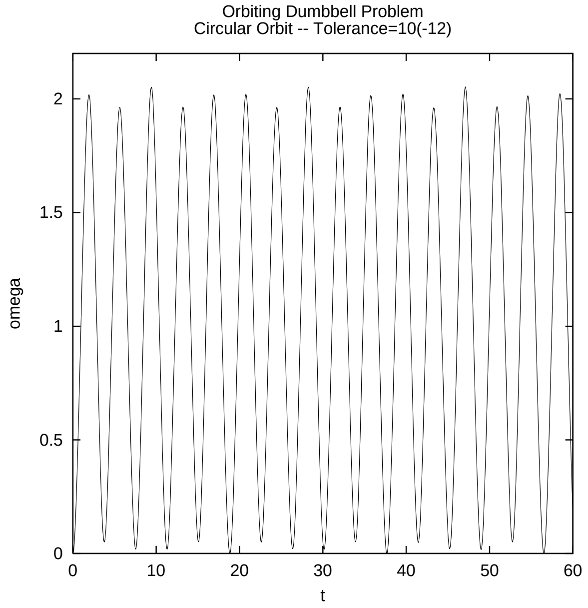


Figure file: `../dumb/om-ez.ps`

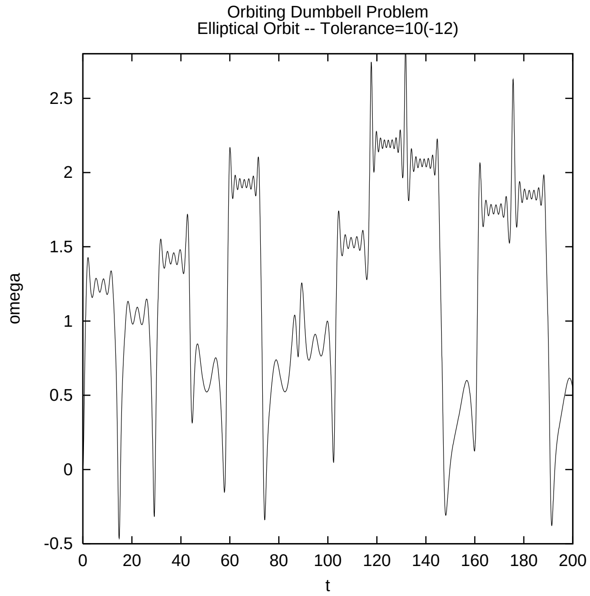


Figure file: `../dumb/om-e.ps`

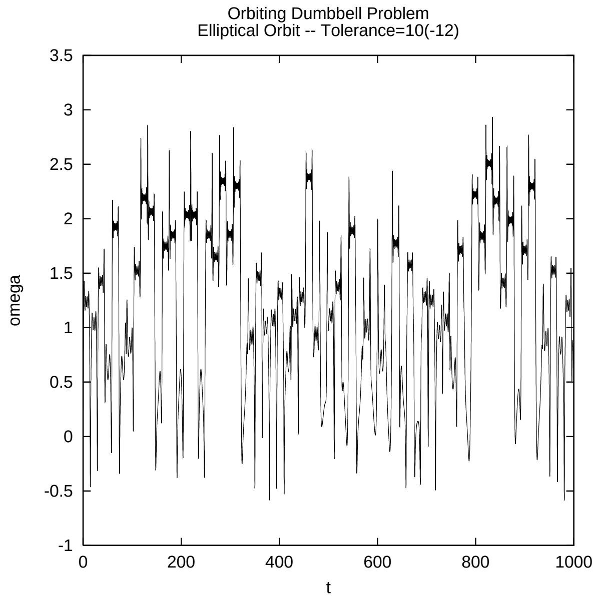


Figure file: `../dumb/e-6.ps`

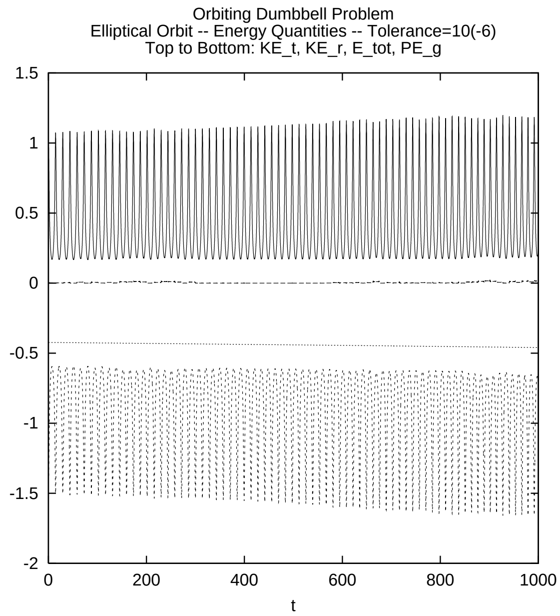


Figure file: `../dumb/e-12.ps`

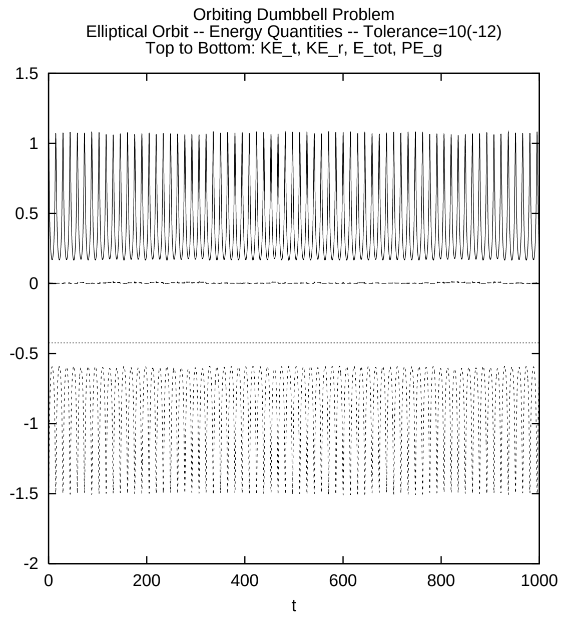
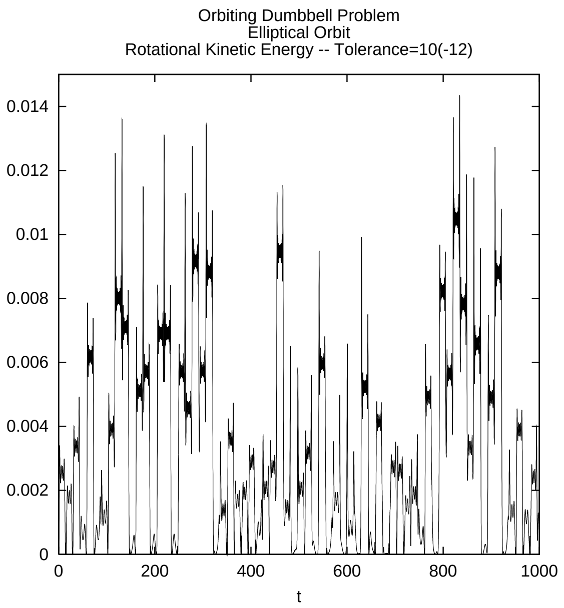


Figure file: `../dumb/kerot-12.ps`



Source file: `wave.f`

```
c===========================================================
c     wave: Solves wave equation:
c
c        u(x,t)_tt = u_xx
c
c     on x = [0..1], t > 0 with initial conditions
c
c        u(x,0)   = exp(-((x-0.5)/0.1)^2)
c        u_t(x,0) = 0
c
c     and boundary conditions
c
c        u(0,t) = u(1,t) = 0
c
c     Solution is obtained using method of lines, with
c     O(h^2) approximation for u_xx, and LSODA to integrate
c     resulting set of ODEs.
c
c     Output is in form suitable for surface-plotting via
c     gnuplot.
c
c     Program also uses 'vsxynt' interface to generate
c     .sdf files which can subsequently be visualized using
c     'jser' (Scivis) and 'sdftosv'.  See Course Sofware
c     page for more details.
c===========================================================
      program        wave

      implicit       none

      integer        iargc,   i4arg
      real*8         r8arg

c-----------------------------------------------------------
c     Include common block for communication with fcn
c-----------------------------------------------------------
      include       'fcn.inc'

c-----------------------------------------------------------
c     Command-line arguments
c-----------------------------------------------------------
      integer        xlevel,     olevel
      real*8         tfinal,     dtout,     tol

c-----------------------------------------------------------
c     Storage for coordinates of spatial mesh and approx.
c     solution.
c-----------------------------------------------------------
      real*8         xmin,           xmax
      parameter    ( xmin = 0.0d0,  xmax = 1.0d0 )
      integer        nxmax
      parameter    ( nxmax = 32 769 )
      real*8         x(nxmax),      y(2 * nxmax)

c-----------------------------------------------------------
c     LSODA declarations
c-----------------------------------------------------------
      external       fcn,        jac
      integer        neq
      real*8         t,          tout
      real*8         rtol,       atol
      integer        itol
      integer        itask,      istate,     iopt
      integer        lrw
      parameter    ( lrw = 22 + 2 * nxmax * 16 )
      real*8         rwork(lrw)
      integer        liw
      parameter    ( liw = 20 + 2 * nxmax )
      integer        iwork(liw)
      integer        jt

c-----------------------------------------------------------
c     Locals.
c-----------------------------------------------------------
      real*8         h,          hm2,        t,
     &               tout
      integer        j,          nx,         stride

c-----------------------------------------------------------
```

```
c     This function, defined in the p410f library, returns
c     its integer argument as a character string.
c-------------------------------------------------------------
      character*2    itoc

c-------------------------------------------------------------
c     Argument parsing and checking.
c-------------------------------------------------------------
      if( iargc() .ne. 5 ) go to 900
      xlevel = i4arg(1,-1)
      olevel = i4arg(2,-1)
      tfinal = r8arg(3,-1.0d0)
      dtout  = r8arg(4,-1.0d0)
      tol    = r8arg(5,-1.0d0)
      if( xlevel .lt. 1  .or.  olevel .lt. 1  .or.
     &    olevel .gt. xlevel  .or. tfinal .lt. 0.0d0  .or.
     &    tol .lt. 0.0d0 ) go to 900

c-------------------------------------------------------------
c     Set up mesh, compute output stride, and initialize
c     mesh coordinates and solution.
c-------------------------------------------------------------
      nx = 2**xlevel + 1
      if( nx .gt. nxmax ) then
         write(0,*) 'wave: Requested nx = ', nx,
     &              'exceeds maximum ', nxmax
         stop
      end if
      stride = 2**(xlevel - olevel)
      h   = (xmax - xmin) / (nx - 1)
      hm2 = 1.0d0 / (h * h)
      x(1)    = xmin
      y(1)    = 0.0d0
      y(1+nx) = 0.0d0
      do j = 2 , nx -1
         x(j)    = x(j-1) + h
         y(j)    = exp( -((x(j) - 0.5d0) / 0.1d0)**2 )
         y(j+nx) = 0.0d0
      end do
      x(nx)    = 1.0d0
      y(nx)    = 0.0d0
      y(nx+nx) = 0.0d0

c-------------------------------------------------------------
c     Set LSODA parameters
c-------------------------------------------------------------
      neq    = 2 * nx
      itol   = 1         ! Indicates that 'atol' is scalar
      rtol   = tol       ! Use same relative and absolute
      atol   = tol       ! tolerances.
      itask  = 1         ! Normal computation
      iopt   = 0         ! Indicates no optional inputs
      jt     = 2         ! Jacobian type

c-------------------------------------------------------------
c     Output initial solution.
c-------------------------------------------------------------
      t = 0.0d0
      call vsxynt('u'//itoc(xlevel),t,x,y,nx)
      call gnuout(y,x,nx,t,stride)
c-------------------------------------------------------------
c     Integrate the approximate solution of the PDE
c     using LSODA.
c-------------------------------------------------------------
      do while( t .lt. tfinal )
         tout = t + dtout
c-------------------------------------------------------------
c        Call lsoda to integrate system on [t ... tout]
c-------------------------------------------------------------
         istate = 1

         call lsoda(fcn,neq,y,t,tout,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
c-------------------------------------------------------------
c        Check return code and exit with error message if
c        there was trouble.
c-------------------------------------------------------------
         if( istate .lt. 0 ) go to 950
c-------------------------------------------------------------
c        Output solution.
c-------------------------------------------------------------
         call vsxynt('u'//itoc(xlevel),t,x,y,nx)
         call gnuout(y,x,nx,t,stride)
      end do

      stop

 900  continue
         write(0,*) 'usage: wave <xlevel> <olevel> '//
     &              '<tfinal> <dtout> <tol>'
      stop

 950  continue
         write(0,*) 'wave: Exiting due to LSODA failure'
         write(0,*) 'wave: Interval ', t, t + dtout
         write(0,*) 'wave: LSODA return code ', istate
      stop

      end

c=============================================================
c     Output to standard out for subsequent plotting via
c     gnuplot.
c=============================================================
      subroutine gnuout(u,x,nx,t,stride)
         implicit       none

         integer        nx,           stride
         real*8         u(nx),        x(nx),        t

         integer        j

         do j = 1 , nx , stride
            write(*,*) t, x(j), u(j)
         end do
         write(*,*)

         return

      end
```

Source file: fcn.f

```
c===========================================================
c     Implements ODEs for method-of-lines solution of
c     wave equation with O(h^2) spatial discretization.
c
c        u_j' = v_j
c        v_j' = hm2 * (v_j+1 - v u_j + v_j-1)
c
c===========================================================
      subroutine fcn(neq,t,y,yprime)
         implicit    none

         include     'fcn.inc'

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

         nx = neq / 2

c-----------------------------------------------------------
c        Dirichlet conditions at x = 0.
c-----------------------------------------------------------
         yprime(1)     = 0.0d0
         yprime(1+nx)  = 0.0d0
         do j = 2 , nx - 1
c-----------------------------------------------------------
c           Interior equations.
c-----------------------------------------------------------
            yprime(j)    = y(j+nx)
            yprime(j+nx) =
     &         hm2 * (y(j+1) - 2.0d0 * y(j) + y(j-1))
         end do
c-----------------------------------------------------------
c        Dirichlet conditions at x = 1.
c-----------------------------------------------------------
         yprime(nx)    = 0.0d0
         yprime(nx+nx) = 0.0d0

         return
      end

c===========================================================
c     Dummy Jacobian routine.
c===========================================================
      subroutine jac
         implicit    none

         return
      end
```

Source file: fcn.inc

```
c===========================================================
c     Common block for communication with 'fcn'
c===========================================================
      real*8                hm2
      common    / com_fcn / hm2
```

Source file: Makefile

```
.IGNORE:

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

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

EXECUTABLES = wave

all: $(EXECUTABLES)

wave.o: wave.f fcn.inc
fcn.o:  fcn.f  fcn.inc

# The libraries '-lsv -lbbhutil -lsv' are needed
# for use of the 'vsxynt' interface.  See Course
# Software page for more details.
wave: wave.o fcn.o
   $(F77_LOAD) wave.o fcn.o \
              -lp410f -lodepack -llinpack -lsvs \
              -lbbhutil -lsv $(LIBBLAS) -o wave

clean:
   rm *.o
   rm $(EXECUTABLES)

vclean: clean
   /bin/rm *.sdf
   /bin/rm *.segdat
   /bin/rm *.ps
   /bin/rm out*
   /bin/ls
```

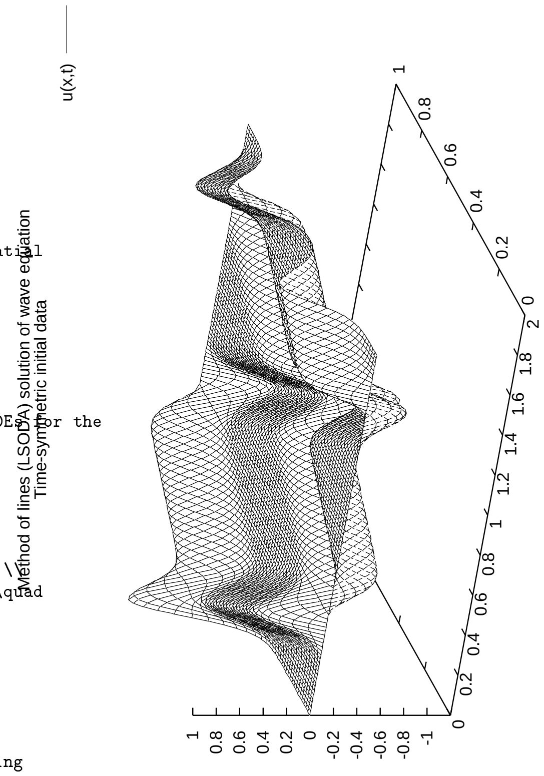
Source file: `Wave`

```
\documentclass [11pt] {article}
\usepackage{verbatim}
\usepackage{epsfig}
%
%
\topmargin=-0.75in
\hoffset=-0.75in
\textheight=9.0in \vsize\textheight
\textwidth=6.5in \hsize\textwidth
\def\lb{\left(}
\def\rb{\right)}
\def\lbb{\left[}
\def\rbb{\right]}
\renewcommand{\baselinestretch}{1.5}
\parindent=0in
\def\S{\vspace*{0.2in}}
%
\begin {document}
\newpage
\setcounter{page}{1}
\setcounter{equation}{0}
{\LARGE \sf
%
\begin{center}
{\bf PHYS 410/555 Computational Physics}\\
{\em The Method of Lines for the Wave Equation}
\end  {center}
%
\vspace*{0.5in}
One approach to the numerical solution of time-dependent {\em partial
differential equations} (PDEs) is to use a discretization technique, such
as finite-differencing, but only apply it explicitly to the
{\em spatial} part(s) of the PDE operator(s) under consideration.
Following the spatial discretization, one is left with a set of
coupled {\em ordinary differential equations} in $t$, which can
then often be solved by a ``standard'' ODE integrator such as
{\tt LSODA}.

\newpage

\S\S
As an example of this technique, consider the {\em wave equation} in
one space dimension (often called the ``1D wave equation):

\begin{equation}
\label{waveA}
\frac{\partial^2}{\partial t^2} u(x,t) =
   c^2 \frac{\partial^2}{\partial x^2} u(x,t)
\end{equation}

\S
Introducing the notation that a subscript denotes partial differentiation,
and suppressing the explicit $x$ and $t$ dependence, (\ref{waveA}) can also
be written as

\begin{equation}
\label{waveb}
u_{tt} = c^2 u_{xx}
\end{equation}

\S
As you probably know, the wave equation describes propagation of
disturbances, or waves, at a speed $c$: waves can either travel
to the right (velocity $+c$), or to the left (velocity $-c$).
Without loss of generality, we can always choose units such that
$c=1$, and, for convenience, we will do so.  Our wave equation then
becomes:

\begin{equation}
\label{waveC}
u_{tt} = u_{xx}
\end{equation}

\newpage
As with any differential equation, boundary conditions play a crucial
role in fixing a solution of~(\ref{waveC}).  Here, we
will solve the wave equation on the domain

\begin{equation}
\label{domain}
0 \le x \le 1 \quad\quad t \ge 0
\end{equation}

\S
and will thus have to provide boundary conditions at $x=0$ and $x=1$, as
well as initial conditions at $t=0$.

For concreteness, we will prescribe {\em Dirichlet boundary conditions}:

\begin{equation}
\label{bcs}
u(0,t) = u(1,t) = 0
\end{equation}

\S
as well as the following {\em initial conditions}:

\begin{eqnarray}
\label{ics}
u(x,0) & = & u_0(x) =
     \exp\left(-\left(\frac{x-x_0}{\Delta}\right)^2\right)\\
\label{icsv}
u_t(x,0) & = & 0
\end{eqnarray}

\S
where $x_0$ ($0 < x_0 < 1$) and $\Delta$ are specified constants.

\newpage
If we think in terms of small-amplitude waves propagating on a string,
then the Dirichlet conditions correspond to keeping the ends of the
string fixed.  The interpretation of the initial conditions is as
follows: In solving~(\ref{waveC}) we have the freedom to specify
the amplitude of the disturbance for all values of $x$, as well as
the time-rate of change of that amplitude, again for all values of
$x$.

\S
We thus have two functions worth of freedom in specifying
our initial conditions.  We set the initial amplitude to some
functional form given by $u_0(x)$; here we use a ``gaussian
pulse'' that is centred at $x_0$, and that has an overall
effective width of a few $\times \,\,\Delta$.
We also set the initial time rate of change of the amplitude to be
0 for all $x$.

\S
Such data is known as {\em time symmetric}, since it
defines an instant in the evolution of the wave equation where there
is a $t \to -t$ symmetry.  In other words, with time symmmetric
initial data, if we integrate {\em backward} in time, we will see
exactly the same solution as a function of $-t$ as we see integrating
forward in time.

\newpage
\S\S
Since the wave equation describes propagating disturbances, and given
that the initial conditions {\em are} time symmetric,
a little reflection might convince you that the
initial conditions~(\ref{ics}) and~ (\ref{icsv}) must represent
a superposition of equal amplitude right-moving and left-moving pulses.
Thus, we should expect the solution of~(\ref{waveC}), subject to~(\ref{bcs}),
(\ref{ics}) and~(\ref{icsv}) to describe the propagation of two equal-amplitude
pulses that are initially coincident, but that subsequently move
apart reflect off $x=0$ and $x=1$ respectively, move together, through
each other, then apart, etc. etc.  Indeed, this is precisely the
behaviour we will observe in our subsequent numerical solution.

\S\S
As mentioned above, the {\em method of lines}, involves an explicit
discretization {\em only of the spatial part of the PDE operator}.
Here we will use the familiar $O(h^2)$ finite-difference approach
to the treatment of $u_{xx} \equiv \partial^2 u / \partial x^2$.

\newpage
However, before proceeding to the spatial discretization, we first
note that~(\ref{waveC}) is a second-order-in-time equation.
In order that our approach eventually produce a set of {\em first order}
ODEs in $t$, we introduce an auxiliary variable, $v(x,t)$,
```

```
\begin{equation}
v(x,t) \equiv u_t(x,t) \equiv \frac{\partial u}{\partial t}(x,t)
\end{equation}

\S
and then rewrite~(\ref{waveC}) as the {\em system:}

\begin{eqnarray}
\label{ut}
u_t & = & v \\
\label{vt}
   v_t & = & u_{xx}
\end{eqnarray}

\S
The boundary conditions become

\begin{equation}
\label{fobcs}
u(0,t) = u(1,t) = v(0,t) = v(1,t) = 0
\end{equation}

\S
while the initial conditions are now

\begin{eqnarray}
\label{foics}
u(x,0) & = & u_0(x) =
      \exp\left(-\left(\frac{x-x_0}{\Delta}\right)^2\right)\\
\label{foicsv}
v(x,0) & = & 0
\end{eqnarray}

\newpage
We can now proceed with the spatial discretization.
To that end, we replace the continuum spatial domain $0 \le x \le 1$
by a uniform finite difference mesh, $x_j$:

\begin{equation}
 x_j \equiv (j - 1) h \quad\quad j = 1, 2, \cdots N \quad\quad\quad
    h \equiv (N - 1)^{-1}
\end{equation}


\S
and introduce the discrete unknowns, $u_j$ and $v_j$:

\begin{eqnarray}
u_j & \equiv & u_j(t) \equiv u(x_j,t) \\
v_j & \equiv & v_j(t) \equiv v(x_j,t)
\end{eqnarray}

\S
Using the usual centred, $O(h^2)$ approximation for the second spatial
derivative,

\begin{equation}
u_{xx}(x_j) = \frac{u_{j+1} - 2 u_j + u_{j-1}}{h^2} + O(h^2)
\end{equation}

\S\S
eqs.~(\ref{ut}) and (\ref{vt}) become a set of $2(N-2)$ coupled ODEs for the
$2(N-2)$ unknowns $u_j(t)$ and $v_j(t)$,~~$j = 2,\cdots N-1$:

\newpage
\begin{eqnarray}
\label{interior}
\frac{du_j}{dt} & = & v_j
   \quad\quad\quad\quad\quad\quad\quad\quad\quad j = 2,\cdots N-1 \\
\frac{dv_j}{dt} & = & \frac{u_{j+1} - 2 u_j + u_{j-1}}{h^2} \quad\quad
   j = 2,\cdots N-1
\end{eqnarray}

\S
We can implement the Dirichlet boundary conditions as follows: if
the boundary conditions are satisfied at the initial time, $t=0$,
then they will be satisfied at all future times provided that
the time derivatives of $u$ and $v$ vanish at the boundaries.  Using
this observation, we
can now write down a complete set of $2 N$ coupled ODEs in the $2 N$
unknowns $u_j(t)$ and $v_j(t)$ which can then be solved using {\tt LSODA}:
```

```
\begin{eqnarray}
\frac{du_1}{dt} &=& 0 \\
\frac{du_j}{dt} &=& v_j \quad\quad\quad\quad\quad\quad\quad\quad\quad
   j = 2,\cdots N-1 \\
\frac{du_N}{dt} &=& 0 \\[0.1in]
\frac{dv_1}{dt} &=& 0 \\
\frac{dv_j}{dt} &=& \frac{u_{j+1} - 2 u_j + u_{j-1}}{h^2} \quad\quad
   j = 2,\cdots N-1 \\
\frac{dv_N}{dt} &=& 0
\end{eqnarray}


\newpage
This solution procedure is implemented by the the program
{\tt wave} (See {\tt $\sim$phys410/ode/wave}).  You will follow
an analagous approach to solve the {\em diffusion equation} in
the final homework.
}
\end {document}
```

Figure file: ../wave/out8.ps



Source file: deut.f

```
c===========================================================
c     deut: Uses LSOSA to integrate ODEs which define a
c     simple model for a deuteron (spherically symmetric,
c     time-independent Schrodinger equation with a square
c     well potential.
c
c     usage: deut <x0> <E> <xmax> <dxout> <tol>
c
c         <x0>     :=  Range of the potential
c         <E>      :=  Estimate of energy eigenvalue, for
c                      any <x0>, there is a single <E0>
c                      which results in a wave function
c                      which -> 0 as <x0> -> infinity.
c         <xmax>   :=  Maximum range of integration
c         <dxout>  :=  Output step. We use this as a
c                      parameter rather than an output
c                      level for ease in extending
c                      integrations to larger <xmax> as
c                      eigenvalue E is determined more
c                      precisely.
c         <tol>    :=  LSODA tolerance parameter.
c
c     Output to standard output is
c
c         <x_j>   <u(x_j)>  <du(x_j)/dx>  <sign(u(x_j))>
c
c     x_j = 0, dxout, 2 dxout, ... xmax
c
c     See class notes and Arfken, Math. Methods for
c     Physicists, 2nd Edition, section 9.1.2
c     for more details.
c===========================================================
      program        deut

      implicit       none

      integer        iargc
      real*8         r8arg

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

c-----------------------------------------------------------
c     Command-line arguments (Note:  x0 and E are defined in
c     fcn.inc)
c-----------------------------------------------------------
      real*8         xmax,       tol

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

      integer        neq
      parameter    ( neq = 2 )

      real*8         y(neq)
      real*8         x,          xout
      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

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

c-----------------------------------------------------------
c     Locals.
c-----------------------------------------------------------
      real*8         dxout

c-----------------------------------------------------------
c     Parse command line arguments.  Deviation from
c-----------------------------------------------------------
      if( iargc() .ne. 5 ) go to 900

      x0      = r8arg(1,r8_never)
      E       = r8arg(2,r8_never)
      xmax    = r8arg(3,r8_never)
      dxout   = r8arg(4,r8_never)
      tol     = r8arg(5,r8_never)
      if( x0   .eq. r8_never  .or.  E     .eq. r8_never  .or.
     &    xmax .eq. r8_never  .or.  dxout .eq. r8_never  .or.
     &    tol  .eq. r8_never ) go to 900

c-----------------------------------------------------------
c     Set LSODA parameters.  Use same value for absolute
c     and relative tolerance.
c-----------------------------------------------------------
      itol   = 1
      rtol   = tol
      atol   = tol
      itask  = 1
      iopt   = 0
      jt     = 2

c-----------------------------------------------------------
c     Initialize the solution, and output it.
c-----------------------------------------------------------
      x    = 0.0d0
      y(1) = 0.0d0
      y(2) = 1.0d0
      write(*,1000) x, y, int(sign(1.0d0,y(1)))
1000  format(1P,3E24.16,0p,i4)

c-----------------------------------------------------------
c     Do the integration.
c-----------------------------------------------------------
      do while( x .lt. xmax )
         istate = 1
         xout   = x + dxout
         call lsoda(fcn,neq,y,x,xout,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)

         if( istate .lt. 0 ) then
            write(0,*) 'deut:  Error return ', istate,
     &                 ' from LSODA '
            write(0,*) 'deut: Current interval ',
     &                 x, x + dxout
            stop
         end if
c-----------------------------------------------------------
c        Output the solution.
c-----------------------------------------------------------
         write(*,1000) x, y, int(sign(1.0d0,y(1)))
      end do

      stop

900   continue
         write(0,*) 'usage: deut <x0> <E> <xmax> '//
     &              '<dxout> <tol>'
      stop

      end
```

Source file: fcn.f

```
c===========================================================
c     Driver routine which integrates ODEs defining
c     model for deuteron.
c
c     See class notes and Arfken, Math. Methods for
c     Physicists, 2nd Edition, section 9.1.2
c     for more details.
c===========================================================
      subroutine fcn(neq,x,y,yprime)
         implicit    none
```

```
         include    'fcn.inc'

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

         real*8      u,     w

         u = y(1)
         w = y(2)
         yprime(1) = w

         if( x .le. x0 ) then
            yprime(2) = (-1.0d0 - E) * y(1)
         else
            yprime(2) = -E * y(1)
         end if

         return
      end

c===========================================================
c     Dummy Jacobian routine.
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
c     x0:    Range of square potential well
c      E:    Energy (sought eigenvalue)
c-----------------------------------------------------------

      real*8
     &         x0,
     &         E
      common / com_fcn /
     &         x0,
     &         E
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = deut

all: $(EXECUTABLES)

deut.o: deut.f fcn.inc

fcn.o:  fcn.f  fcn.inc

deut: deut.o fcn.o
   $(F77_LOAD) deut.o fcn.o -lp410f -lodepack \
               -llinpack $(LIBBLAS) -o deut

clean:
   /bin/rm $(EXECUTABLES)
   /bin/rm *.o
```

Source file: `Shoot-deut`

```
#!/bin/sh -x

############################################################
# Computes eigenvalue E = E(x0) for toy-deuteron problem
# using "shooting" and bisection search. Uses the following
# empirical facts:
#
#   If E_trial > E then u(xmax) > 0
#   If E_trial < E then u(xmax) < 0
#
# Uses perl scripts
#
#   bsnew
#   bslo
#   bshi
#   bsdone
#
# which provide rudimentary bisection search facility
#
# An initial bracket [<Elo>,<Ehi>] must be provided, as well
# as a tolerance <Etol> for the bisection search.
#
# Output accumulated in files
# x0=<x0>/E=<E>
############################################################

P=`basename $0`

usage() {
printf "$P <x0> <Elo> <Ehi> <Etol> <xmax> <dxout>"
printf " <lsoda tol> [<vstrace>]\n"
exit 1
}

die() {
echo "$P $1"
exit 1
}

case $# in
7|8) x0=$1; Elo=$2; Ehi=$3; Etol=$4; xmax=$5; dxout=$6;
   lsodatol=$7; vstrace=${8-false};
   case $vstrace in
   true|false) ;;
   *) "vstrace must be 'true' or 'false'";;
   esac;;
*) usage;;
esac

# Create results directory if necessary
dir="x0=$x0"
test -d $dir || mkdir $dir

# Initialize the bisection search
bsnew $Elo $Ehi

# Perform the bisection search
while bsnotdone; do
   Ecurr=`bscurr`
   ofile="$dir/E=$Ecurr"
   deut $x0 $Ecurr $xmax $dxout $lsodatol > $ofile
   $vstrace && nth 1 2 < $ofile | vn $P $x0
   flag=`tail -1 $ofile | nth 4`
   case $flag in
    1) bshi;;
   -1) bslo;;
    *) echo "$P: Unexpected flag value '$flag'"; exit 1;;
   esac
done
nth 1 2 < $ofile > $dir/solution

printf "%12s %25s %25s %12s %12s\n" \
   $x0 $Ecurr `bsfrac` $dxout $lsodatol >> deut-results
```

Source file: `mkplots`

```
#!/bin/sh -x
P=`basename $0`

#------------------------------------------------------------
# mkplots: script for plotting results from 'deut'
#------------------------------------------------------------
die() {
echo "$P: $1"
exit 1
}

for x0 in 2.0 4.0 6.0 8.0; do
   dir="x0=$x0"
   test -d $dir || die "Directory '$dir' does not exist"
   sfile="$dir/solution"
   test -f $sfile || \
      die "Solution file '$sfile' does not exist"
done

test -f u.ps || gnuplot<<END
set terminal postscript portrait
set output "u.ps"
set size square
set title "Toy Model Deuteron Wave Functions\nUnit \
depth square-well potential with range x0\n(Wave \
functions are unnormalized)"
set xlabel "x"
set ylabel "u(x)"
plot [0:20] [0:3] \
     "x0=2.0/solution" title "x0=2.0" with lines, \
     "x0=4.0/solution" title "x0=4.0" with lines, \
     "x0=6.0/solution" title "x0=6.0" with lines, \
     "x0=8.0/solution" title "x0=8.0" with lines
quit
END

dir="x0=2.0-detail"
test -d $dir || mkdir $dir
cd $dir

for E in -0.0900 -0.1100 -0.1000 -0.1050 -0.1025; do
   test -f E=$E || \
      deut 2.0 $E 30.0 0.01 1.0d-10 | nth 1 2 > E=$E
done

test -f ../shoot.ps || gnuplot<<END
set terminal postscript portrait
set output "shoot.ps"
set size square
set title "Toy Model Deuteron Wave Functions\nUnit depth \
square-well potential with range x0\nIllustration of \
bisection solution for eigenvalue"
set xlabel "x"
set ylabel "u(x)"
plot [0:30] [-40:40] \
     "E=-0.0900" title "E=-0.0900" with lines, \
     "E=-0.1100" title "E=-0.1100" with lines, \
     "E=-0.1000" title "E=-0.1000" with lines, \
     "E=-0.1050" title "E=-0.1050" with lines, \
     "E=-0.1025" title "E=-0.1025" with lines
quit
END

test -f ../zshoot.ps || gnuplot<<END
set terminal postscript portrait
set output "zshoot.ps"
set size square
set title "Toy Model Deuteron Wave Functions\nUnit depth \
square-well potential with range x0\nIllustration of \
bisection solution for eigenvalue (detail)"
set xlabel "x"
set ylabel "u(x)"
plot [0:10] [0:1.2] \
     "E=-0.0900" title "E=-0.0900" with lines, \
     "E=-0.1100" title "E=-0.1100" with lines, \
     "E=-0.1000" title "E=-0.1000" with lines, \
     "E=-0.1050" title "E=-0.1050" with lines, \
     "E=-0.1025" title "E=-0.1025" with lines
quit
END

ls *.ps > /dev/null 2>&1 && mv *.ps ..
cd ..

ls -lt *ps
```

Figure file: `../deut/shoot.ps`

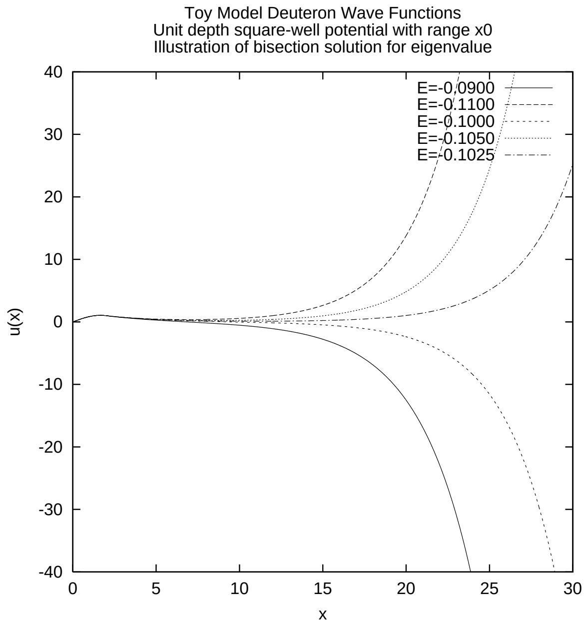


Figure file: `../deut/zshoot.ps`

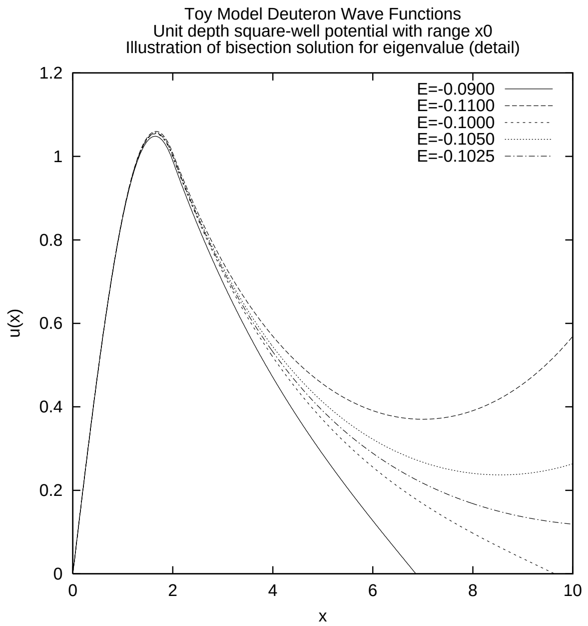


Figure file: `../deut/u.ps`