Source file: `lsoda.f (header)`

```
      subroutine lsoda (f, neq, y, t, tout, itol, rtol, atol, itask,
     1            istate, iopt, rwork, lrw, iwork, liw, jac, jt)
      external f, jac
      integer neq, itol, itask, istate, iopt, lrw, iwork, liw, jt
      double precision y, t, tout, rtol, atol, rwork
      dimension neq(1), y(1), rtol(1), atol(1), rwork(lrw), iwork(liw)
c-----------------------------------------------------------------------
c this is the march 30, 1987 version of
c lsoda.. livermore solver for ordinary differential equations, with
c         automatic method switching for stiff and nonstiff problems.
c
c this version is in double precision.
c
c lsoda solves the initial value problem for stiff or nonstiff
c systems of first order ode-s,
c     dy/dt = f(t,y) ,  or, in component form,
c     dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(neq)) (i = 1,...,neq).
c
c this a variant version of the lsode package.
c it switches automatically between stiff and nonstiff methods.
c this means that the user does not have to determine whether the
c problem is stiff or not, and the solver will automatically choose the
c appropriate method.  it always starts with the nonstiff method.
c
c authors..
c               linda r. petzold  and  alan c. hindmarsh,
c               computing and mathematics research division, l-316
c               lawrence livermore national laboratory
c               livermore, ca 94550.
c
c references..
c 1.  alan c. hindmarsh,  odepack, a systematized collection of ode
c     solvers, in scientific computing, r. s. stepleman et al. (eds.),
c     north-holland, amsterdam, 1983, pp. 55-64.
c 2.  linda r. petzold, automatic selection of methods for solving
c     stiff and nonstiff systems of ordinary differential equations,
c     siam j. sci. stat. comput. 4 (1983), pp. 136-148.
c-----------------------------------------------------------------------
c summary of usage.
c
```

.
.
.

Source file: `tlsoda.f`

```
c===========================================================
c     tlsoda: Demo program which uses ODEPACK routine LSODA
c     to solve the second-order ODE
c
c        u''(t) = -u(t),  0 <= t <= tmax
c
c     (' = d/dt), with initial conditions
c
c        u(0) = u0,   u'(0) = du0
c
c-----------------------------------------------------------
c     usage: tlsoda <tmax> <u0> <du0> <tol> <olevel>
c-----------------------------------------------------------
c
c     The exact solution is
c
c        u_xct(t) = du0 * sin(t) + u0 * cos(t)
c
c     Output to standard out is
c
c        t_it    u_it   [u_xct - u]_it
c
c     it = 1, 2, ... nout, where
c
c        t_1     = 0
c        t_ntout = tmax
c        ntout   = 2**olevel + 1
c
c     Output to standard error is the RMS error of the
c     approximate solution.
c===========================================================
      program        tlsoda

      implicit       none

      integer        iargc,        i4arg
      real*8         r8arg

c-----------------------------------------------------------
c     Command-line arguments:
c
c     tmax:    Final integration time
c     u0:      Initial value: u(0)
c     du0:     Initial value: u'(0)
c     tol:     Error tolerance (this program uses LSODA's
c              pure absolute error control)
c     olevel:  Output level:  dtout = tmax/2**olevel
c-----------------------------------------------------------
      real*8         tmax,       u0,      du0,     tol
      integer        olevel
      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )

c===========================================================
c     Start of LSODA declarations
c===========================================================

c-----------------------------------------------------------
c     Note that 'fcn' and 'jac' are user supplied SUBROUTINES
c     (not functions) which evaluate the RHSs of the ODEs and
c     the Jacobian of the system.  Under normal operation,
c     (as in this case), the Jacobian evaluator can be a
c     'dummy' routine; if and when needed, LSODA will compute
c     a finite-difference approximation to the Jacobian.
c-----------------------------------------------------------
      external       fcn,        jac
c-----------------------------------------------------------
c     Number of ODEs (when written in canonical first order
c     form).
c-----------------------------------------------------------
      integer        neq
      parameter     ( neq = 2 )
c-----------------------------------------------------------
c     y(neq): Storage for approximate solution
c     t:      Initial time for LSODA integration sub-interval
c     tout:   Final time for LSODA integration sub-interval
c-----------------------------------------------------------
      real*8         y(neq)
      real*8         t,          tout

c-----------------------------------------------------------
c     Tolerance parameters:
c
c     The following comment block is extracted from the
c     LSODA documentation.
c-----------------------------------------------------------
c rtol   = relative tolerance parameter (scalar).
c atol   = absolute tolerance parameter (scalar or array).
c  the estimated local error in y(i) will be controlled so
c  as to be less than
c       ewt(i) = rtol*abs(y(i)) + atol     if itol = 1, or
c       ewt(i) = rtol*abs(y(i)) + atol(i)  if itol = 2.
c  thus the local error test passes if, in each component,
c  either the absolute error is less than atol (or atol(i)),
c  or the relative error is less than rtol.
c  use rtol = 0.0 for pure absolute error control, and
c  use atol = 0.0 (or atol(i) = 0.0) for pure relative error
c  control.  CAUTION.. actual (global) errors may exceed
c  these local tolerances, so choose them CONSERVATIVELY.
c-----------------------------------------------------------
      real*8        rtol,      atol
      integer       itol

c-----------------------------------------------------------
c     Control parameters and return code (see below).
c-----------------------------------------------------------
      integer       itask,      istate,     iopt

c-----------------------------------------------------------
c     Work arrays.
c-----------------------------------------------------------
      integer       lrw
      parameter   ( lrw = 22 + neq * 16 )
      real*8        rwork(lrw)

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

c-----------------------------------------------------------
c     'jt' defines which type of Jacobian is supplied or
c     computed; we use jt = 2 here which, as mentioned
c     above, instructs LSODA to compute a finite-difference
c     approximation to the Jacobian if and when needed.
c-----------------------------------------------------------
      integer       jt

c===========================================================
c     End of LSODA declarations
c===========================================================

c-----------------------------------------------------------
c     Miscellaneous variables
c-----------------------------------------------------------
      real*8         dtout,    err,     rmserr
      integer        it,       ntout

c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .ne. 5 )  go to 900
      tmax   = r8arg(1,r8_never)
      u0     = r8arg(2,r8_never)
      du0    = r8arg(3,r8_never)
      tol    = r8arg(4,r8_never)
      olevel = i4arg(5,-1)
      if( tmax .eq. r8_never .or.  u0  .eq. r8_never .or.
     &    du0  .eq. r8_never .or.  tol .eq. r8_never .or.
     &    olevel .lt. 0 )
     &  go to 900

c-----------------------------------------------------------
c     Set LSODA parameters ... see LSODA documentation
c     for fuller description.
c-----------------------------------------------------------
      itol   = 1        ! Indicates that 'atol' is scalar
      rtol   = 0.0d0    ! Use pure absolute tolerance
      atol   = tol      ! Absolute tolerance
      itask  = 1        ! Normal computation
      iopt   = 0        ! Indicates no optional inputs
      jt     = 2        ! Jacobian type
```

```
c-----------------------------------------------------------
c     Compute number of output times and output interval,
c     and intialize sub-interval start time and solution
c     estimate.
c-----------------------------------------------------------
      ntout  = 2**olevel + 1
      dtout  = tmax / (ntout - 1)
      t      = 0.0d0
      y(1)   = u0
      y(2)   = du0

c-----------------------------------------------------------
c     Output initial solution and error and initialize
c     rms error.
c-----------------------------------------------------------
      err = du0 * sin(t) + u0 * cos(t) - y(1)
      write(*,*) t, y(1), err
      rmserr = err**2

c-----------------------------------------------------------
c     Loop over requested output times ...
c
c     Set istate to 1 to indicate initial call, istate
c     should be set to 2 for subsequent calls, but lsoda
c     will automatically do this so long as the initial
c     call is successful.
c-----------------------------------------------------------
      istate = 1

      do it = 2, ntout
c-----------------------------------------------------------
c        Set final integration time for current interval ...
c-----------------------------------------------------------
         tout = t + dtout
c-----------------------------------------------------------
c        Call lsoda to integrate system on [t ... tout]
c
c        Note that LSODA replaces 't' with the value
c        of 'tout' if the integration is successful.
c-----------------------------------------------------------

         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 ) then
            write(0,1000) istate, it, ntout, t, t + dtout
1000        format(/' sode: Error return ',i2,
     &              ' from integrator LSODA.'/
     &              ' sode: At output time ',i5,' of ',i5/
     &              ' sode: Interval ',1p,e11.3,0p,
     &              ' .. ',1p,e11.3,0p/)
            go to 500
         end if
c-----------------------------------------------------------
c        Output the solution and error, and update RMS error
c        accumulator.
c-----------------------------------------------------------
         err = du0 * sin(t) + u0 * cos(t) - y(1)
         write(*,*) t, y(1), err
         rmserr = rmserr + err**2
      end do
c-----------------------------------------------------------
c     Output the RMS error to standard error.
c-----------------------------------------------------------
      rmserr = sqrt(rmserr / ntout)
      write(0,*) 'rmserr: ', rmserr

 500  continue

      stop

 900  continue
         write(0,*) 'usage: tlsoda <tmax> <u0> <du0> '//
     &              '<tol> <olevel>'
      stop

      end

c===========================================================
c     Implements differential equations:
c
c     u'' = -u
c
c     y(1) := u
c     y(2) := u'
c
c     y(1)' :=  y(2)
c     y(2)' := -y(1)
c
c     Called by ODEPACK routine LSODA.
c===========================================================
      subroutine fcn(neq,t,y,yprime)
         implicit    none

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

         yprime(1) =  y(2)
         yprime(2) = -y(1)

         return
      end

c===========================================================
c     Implements Jacobian (optional).  Dummy routine in
c     this case.
c===========================================================
      subroutine jac
         implicit    none

         return
      end
```

Source file: chk-tlsoda.f

```
c===========================================================
c     chk_tlsoda:  Program to check the output of tlsoda
c     by applying a second-order discretization of the ODE
c     to the computed solution.
c
c     Output is dt and the RMS value of the residual of the
c     O(dt^2) discretization, which should itself be
c     approximately O(dt^2); refer to class notes for more
c     details.
c===========================================================
      program       chk_tlsoda

      implicit      none

      integer       maxnt
      parameter   ( maxnt = 100 000 )

      real*8        t(maxnt),     u(maxnt)
      real*8        hm2,          rmsres
      integer       nt,           it

      call dvvfrom('-',t,u,nt,maxnt)
c-----------------------------------------------------------
c     Will assume that 't' defines a *uniform* mesh.
c-----------------------------------------------------------
      hm2 = 1.0d0 / (t(2) - t(1))**2
      rmsres = 0.0d0
      do it = 2 , nt - 1
         rmsres = rmsres +
     &      ( hm2 * (u(it+1) - 2.0d0 * u(it) + u(it-1)) +
     &        u(it) )** 2
      end do
      rmsres = sqrt(rmsres / (nt - 2))
      write(*,*) t(2) - t(1), rmsres

      stop

900   continue
         write(0,*) 'usage: chk_tsloda'
         write(0,*) ' '
         write(0,*) '      Reads (x_i, u_i) pairs from '//
```

```
&                        'standard input'
 stop

 end
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = tlsoda chk-tlsoda

all: $(EXECUTABLES)

# Note that usage of 'odepack' library also requires linking to
# 'linpack' library (LINPACK is an antecedent of LAPACK)

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

chk-tlsoda: chk-tlsoda.o dvvfrom.o
   $(F77_LOAD) chk-tlsoda.o dvvfrom.o -lp410f -o chk-tlsoda

clean:
   rm *.o
   rm $(EXECUTABLES)

vclean: clean
   rm tlsoda-out*
   rm tlsoda-u*
   rm tlsoda-err*
   rm *.ps
```

Source file: `Tlsoda`

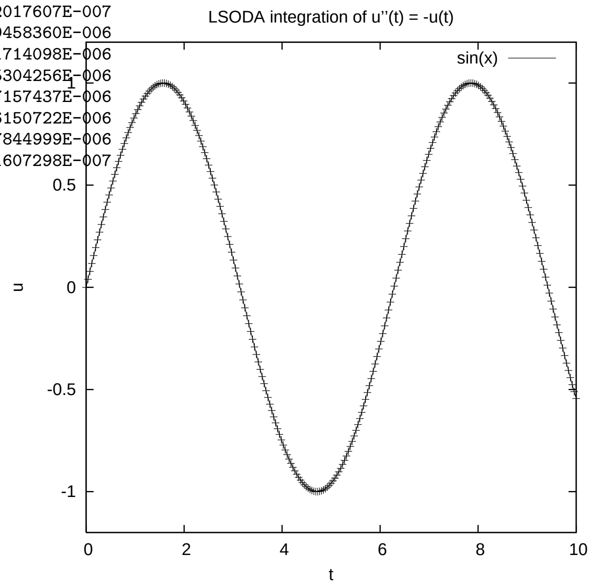
```
#! /bin/sh

############################################################
# Script which runs 'tlsoda' with a variety of tolerance
# settings, checks one solution using "independent residual
# evaluation", and demonstrates dependence of results on
# number of requested output times.
############################################################

# Integrate from 0 .. 10
tmax=10.0

# Exact solution is sin(t)
utmin=0.0
dutmin=1.0

olevel=8
tols="1.0e-6 1.0e-8 1.0e-10 1.0e-12"

# Make sure executable exists, make if it isn't
test -f tlsoda || make

echo "---------------------------------------------"
echo "Running tlsoda with the following tolerances:";
echo "  $tols"
echo "---------------------------------------------"
for tol in $tols; do
   echo "Tolerance: $tol";
   tlsoda $tmax $utmin $dutmin $tol $olevel > tlsoda-out-$tol
#  Create file with column 1=x column 2=u
   nth 1 2 < tlsoda-out-$tol > tlsoda-u-$tol
#  Create file with column 1=x column 2=abs(u_xct - u)
   nth 1 3 < tlsoda-out-$tol | nf _1 'abs(_2)' > tlsoda-err-$tol
done
echo

checktol="1.0e-12"
echo "---------------------------------------------"
echo "Applying O(dt^2) approximation of ODE to "
echo "tol=$checktol results"
echo "---------------------------------------------"
echo "          dt                    rms(residual)"
for inc in 8 4 2 1; do
#  'lines' is a filter which selects line-number ranges
```

```
   nth 1 2 < tlsoda-out-$checktol | lines 1 . $inc | chk-tlsoda
done
echo

echo "--------------------------------------------"
echo "Demonstrating dependence of results on number"
echo "of requested output times"
echo "--------------------------------------------"
for tol in $tols; do
   echo "Tolerance: $tol";
   echo "No additional output times"
   tlsoda $tmax $utmin $dutmin $tol 0 > /dev/null
   echo "256 output times"
   tlsoda $tmax $utmin $dutmin $tol 8 > /dev/null
   echo "65536 output times"
   tlsoda $tmax $utmin $dutmin $tol 16 > /dev/null
   echo
done

# Make plots of soln and error
gnuplot < gpin
gnuplot < gpine
```

Source file: Output from Tlsoda on lnx1

```
############################################################
# Demonstration of use of 'tlsoda' and 'chk-tlsoda'
############################################################

% pwd; ls
/home/phys410/ode/tlsoda
Makefile  Tlsoda*  chk-tlsoda.f  dvvfrom.f  gpin  gpine  tlsoda.f

% make
pgf77 -g -c tlsoda.f
pgf77 -g -L/usr/local/PGI/lib tlsoda.o -lp410f \
               -lodepack -llinpack -lblas -o tlsoda
pgf77 -g -c chk-tlsoda.f
pgf77 -g -c dvvfrom.f
pgf77 -g -L/usr/local/PGI/lib chk-tlsoda.o \
               dvvfrom.o -lp410f -o chk-tlsoda

% tlsoda
 usage: tlsoda <tmax> <u0> <du0> <tol> <olevel>

% tlsoda 1.0 0.0 2.0 1.0d-6 3
   0.0000000000000000E+000  0.0000000000000000E+000  0.0000000000000000E+000
   0.1250000000000000       0.2493503338815517      -8.6711109632017607E-007
   0.2500000000000000       0.4948090295080188      -1.1109989729458360E-006
   0.3750000000000000       0.7325463645140672      -1.3063419721714098E-006
   0.5000000000000000       0.9588526174206525      -1.5402122465304256E-006
   0.6250000000000000        1.170196263557266      -1.7176763417157437E-006
   0.7500000000000000        1.363279629007695      -2.1089610266150722E-006
   0.8750000000000000        1.535088083213516      -1.0787414617844999E-006
    1.000000000000000        1.682942631747096      -6.6213130311607298E-007
 rmserr:   1.2937977308201230E-006

############################################################
# Invoke 'Tlsoda' script to put 'tlsoda' through its paces
############################################################

% Tlsoda
---------------------------------------------
Running tlsoda with the following tolerances:
  1.0e-6 1.0e-8 1.0e-10 1.0e-12
---------------------------------------------
Tolerance: 1.0e-6
 rmserr:   5.4901404035008383E-006
Tolerance: 1.0e-8
 rmserr:   1.1801577890437745E-008
Tolerance: 1.0e-10
 rmserr:   5.6420426412897609E-010
Tolerance: 1.0e-12
 rmserr:   7.4219223237344245E-012

---------------------------------------------
Applying O(dt^2) approximation of ODE to
tol=1.0e-12 results
---------------------------------------------
          dt                    rms(residual)
   0.3125000000000000       5.6697977613005648E-003
   0.1562500000000000       1.4121956372318181E-003
   7.8125000000000000E-002  3.5224715816071033E-004
   3.9062500000000000E-002  8.7952495474958567E-005

---------------------------------------------
Demonstrating dependence of results on number
of requested output times
---------------------------------------------
Tolerance: 1.0e-6
No additional output times
 rmserr:   1.1209185226999998E-005
256 output times
 rmserr:   5.4901404035008383E-006
65536 output times
 rmserr:   5.0924090121097052E-006

Tolerance: 1.0e-8
No additional output times
 rmserr:   5.4600515843744528E-009
256 output times
 rmserr:   1.1801577890437745E-008
65536 output times
 rmserr:   2.7422522181745856E-008

Tolerance: 1.0e-10
No additional output times
 rmserr:   9.8276284230249562E-010
256 output times
 rmserr:   5.6420426412897609E-010
65536 output times
 rmserr:   6.0607617739567266E-010

Tolerance: 1.0e-12
No additional output times
 rmserr:   3.8023204681436628E-012
256 output times
 rmserr:   7.4219223237344245E-012
65536 output times
 rmserr:   6.3700996654287074E-012
```

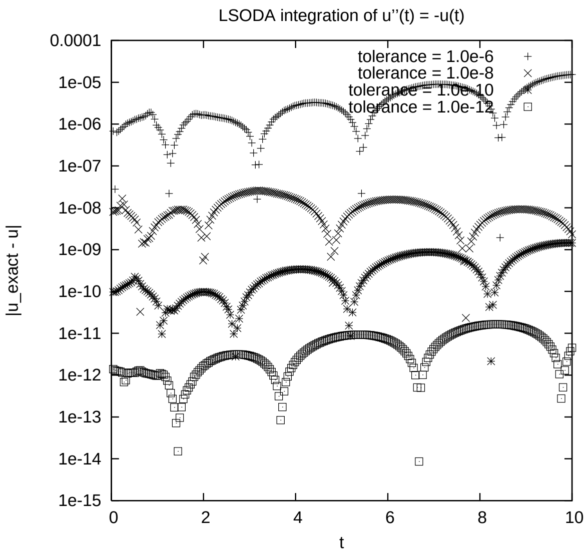
Figure file: ../tlsoda/soln.ps



Source file: `gnuplot commands for soln.ps`

```
set terminal postscript portrait
set output "soln.ps"
set size square
set title "LSODA integration of u''(t) = -u(t)"
set xlabel "t"
set ylabel "u"
plot [0:10] [-1.2:1.2] sin(x),  "tlsoda-u-1.0e-6" notitle
quit
```

Figure file: `../tlsoda/error.ps`



Source file: `gnuplot commands for error.ps`

```
set terminal postscript portrait
set output "error.ps"
set size square
set title "LSODA integration of u''(t) = -u(t)"
set xlabel "t"
set ylabel "|u_exact - u|"
set nologscale; set logscale y
plot "tlsoda-err-1.0e-6"  title 'tolerance = 1.0e-6', \
     "tlsoda-err-1.0e-8"  title 'tolerance = 1.0e-8', \
     "tlsoda-err-1.0e-10" title 'tolerance = 1.0e-10', \
     "tlsoda-err-1.0e-12" title 'tolerance = 1.0e-12'
quit
```

Source file: `Utility commands`

```
#############################################################
# Illustrates use of some utility commands available
# on sgi1, vnfe1 and lnx[123] (but note that 'paste' is a
# generic Unix command) which are useful for generating and
# manipulating columns of numbers.
#
# (1) dvmesh: Generates uniform sequence of real numbers.
#     Included here mostly as a mechanism to generate
#     input for 'nf'.  (Instructor-supplied C-program).
#
# (2) nf: Generalization of 'nth'. (See Course Notes for
#     October 5th.)  Filter which selects columns from
#     standard input (assumed numeric), performs fairly
#     general mathematical operations as needed, and outputs
#     one or more columns of numbers on standard output.
#     (Instructor supplied perl-script).
#
# (3) paste: Standard Unix facility for combining ('pasting')
#     one or more file arguments, see
#
#        man paste
#
#     for more information, BUT note that I typically use the
#     alias
#
#        alias paste 'paste -d" "'
#
#     so that paste uses a blank (space) rather than <TAB>
#     as the catenation character.  For the purposes of
#     the course, the two types of invocation should be
#     equivalent, and I have NOT set up your accounts on
#     'sgi1' so that the above alias is defined by default.
#     Recall that, in a C-shell, you can always find out exactly
#     which command a particular command-name will expand to
#     using 'which':
#
#        sgi% which paste
#        paste:   aliased to paste -d" "
#
#        sgi% unalias paste
#
#        sgi% which paste
#        /usr/bin/paste
############################################################

############################################################
# Usage of 'dvmesh' is straightforward.  The command
# generates a length 'n' sequence of real numbers, uniformly
# spaced, and ranging from 'xmin' to 'xmax'.
############################################################

sgi1% dvmesh
usage: dvmesh <xmin> <xmax> <n > 0>

sgi1% dvmesh 0.0 1.0 11
  0.0000000000000000E+00
  1.0000000000000001E-01
  2.0000000000000001E-01
  3.0000000000000004E-01
  4.0000000000000002E-01
  5.0000000000000000E-01
  5.9999999999999998E-01
  6.9999999999999996E-01
  7.9999999999999993E-01
  8.9999999999999991E-01
  1.0000000000000000E+00


############################################################
# 'nf' accepts an arbitrary number of arguments, reads
# columns of numbers from standard input, then manipulates
# the input-columns and writes the results to standard
# output.  Use the notation '_1', '_2' etc. to refer to
# the first, second etc. column.  Usage is best demonstrated
# with some examples:
############################################################

sgi1% nf
usage: nf <expr #> [<expr #> ...]

############################################################
# Compute x^2, x = 0.0, 0.1, ... 0.9,  1.0 and write
# (x, x^2) to standard output.  Note use of single quotes
# around 2nd argument to 'nf' to inhibit shell-interpretation
# of multiplication operator '*'.
############################################################

sgi1% dvmesh 0.0 1.0 11 | nf _1 '_1 * _1'
0.0000000000000000E+00   0
1.0000000000000001E-01   0.01
2.0000000000000001E-01   0.04
3.0000000000000004E-01   0.09
4.0000000000000002E-01   0.16
5.0000000000000000E-01   0.25
5.9999999999999998E-01   0.36
6.9999999999999996E-01   0.49
7.9999999999999993E-01   0.64
8.9999999999999991E-01   0.81
```

```
1.0000000000000000E+00   1



############################################################
# Repeat the calculation and redirect to a file 'squares'.
# Compute the cubes of the same x-values and redirect
# (x,x^3) to 'cubes'.
############################################################

sgi1% dvmesh 0.0 1.0 11 | nf _1 '_1 * _1' > squares
sgi1% cat squares
0.0000000000000000E+00   0
1.0000000000000001E-01   0.01
2.0000000000000001E-01   0.04
3.0000000000000004E-01   0.09
4.0000000000000002E-01   0.16
5.0000000000000000E-01   0.25
5.9999999999999998E-01   0.36
6.9999999999999996E-01   0.49
7.9999999999999993E-01   0.64
8.9999999999999991E-01   0.81
1.0000000000000000E+00   1

sgi1% dvmesh 0.0 1.0 11 | nf _1 'pow(_1,3)' > cubes
sgi1% cat cubes
0.0000000000000000E+00   0
1.0000000000000001E-01   0.001
2.0000000000000001E-01   0.008
3.0000000000000004E-01   0.027
4.0000000000000002E-01   0.064
5.0000000000000000E-01   0.125
5.9999999999999998E-01   0.216
6.9999999999999996E-01   0.343
7.9999999999999993E-01   0.512
8.9999999999999991E-01   0.729
1.0000000000000000E+00   1

############################################################
# 'nf' understands
#
# (A) The usual binary arithmetic operations: *, /, +, -,
# (B) Integer power function (uses repeated multiplies)
#     ipow(ix,iy) = ix^iy
# (C) Real power function (uses logs and exponentiation)
#     pow(x,y) = x^y (x must be postive-definite)
# (D) min() and max() of an arbitrary number of arguments
# (E) The usual suite of mathematical functions: sin, cos,
#     tan, sinh, cosh, tanh, exp, log, abs, sqrt (inverse
#     trig and hyperbolic function are currently *not*
#     implemented.)
############################################################

sgi1% dvmesh 0.0 4.0 11 | nf _1 'sin(_1)' 'cos(_1)' \
?         'ipow(sin(_1),2) + ipow(cos(_1),2)'
0.0000000000000000E+00   0   1   1
4.0000000000000002E-01   0.389418342308651   0.921060994002885   1
8.0000000000000004E-01   0.717356090899523   0.696706709347165   1
1.2000000000000002E+00   0.932039085967226   0.362357754476673   1
1.6000000000000001E+00   0.999573603041505   -0.0291995223012889   1
2.0000000000000000E+00   0.909297426825682   -0.416146836547142   1
2.3999999999999999E+00   0.675463180551151   -0.737393715541246   1
2.7999999999999998E+00   0.334988150155905   -0.942222340668658   1
3.1999999999999997E+00   -0.0583741434275798   -0.998294775794753   1
3.5999999999999996E+00   -0.442520443294852   -0.896758416334147   1
4.0000000000000000E+00   -0.756802495307928   -0.653643620863612   1

############################################################
# 'paste': Combines files 'horizontally' in a straightforward
# fashion.  Most useful for use with two or more files each
# of which contain one or more columns with, but
# which all contain the same number of lines (length of
# columns).  Note that paste's output is to standard out.
############################################################

sgi1% paste squares cubes
0.0000000000000000E+00   0      0.0000000000000000E+00   0
1.0000000000000001E-01   0.01   1.0000000000000001E-01   0.001
2.0000000000000001E-01   0.04   2.0000000000000001E-01   0.008
3.0000000000000004E-01   0.09   3.0000000000000004E-01   0.027
4.0000000000000002E-01   0.16   4.0000000000000002E-01   0.064
5.0000000000000000E-01   0.25   5.0000000000000000E-01   0.125
5.9999999999999998E-01   0.36   5.9999999999999998E-01   0.216
6.9999999999999996E-01   0.49   6.9999999999999996E-01   0.343
7.9999999999999993E-01   0.64   7.9999999999999993E-01   0.512
8.9999999999999991E-01   0.81   8.9999999999999991E-01   0.729
1.0000000000000000E+00   1      1.0000000000000000E+00   1

############################################################
# The above is probably not quite what we wanted.  Use
# 'nf' (or 'nth') to get rid of third column.  Note that
# 'nth' refers to columnm 1, 2 etc simply as '1', '2'.
############################################################

sgi1% paste squares cubes | nf _1 _2 _4
0.0000000000000000E+00   0   0
1.0000000000000001E-01   0.01   0.001
2.0000000000000001E-01   0.04   0.008
3.0000000000000004E-01   0.09   0.027
4.0000000000000002E-01   0.16   0.064
5.0000000000000000E-01   0.25   0.125
5.9999999999999998E-01   0.36   0.216
6.9999999999999996E-01   0.49   0.343
7.9999999999999993E-01   0.64   0.512
8.9999999999999991E-01   0.81   0.729
1.0000000000000000E+00   1   1

sgi1% paste squares cubes | nth 1 2 4
0.0000000000000000E+00 0 0
1.0000000000000001E-01 0.01 0.001
2.0000000000000001E-01 0.04 0.008
3.0000000000000004E-01 0.09 0.027
4.0000000000000002E-01 0.16 0.064
5.0000000000000000E-01 0.25 0.125
5.9999999999999998E-01 0.36 0.216
6.9999999999999996E-01 0.49 0.343
7.9999999999999993E-01 0.64 0.512
8.9999999999999991E-01 0.81 0.729
1.0000000000000000E+00 1 1
```

Source file: `integral.f`

```
c===========================================================
c     Program demonstrating use of 'lsoda' to evaluate
c     a definite integral.
c
c     Also demonstrates use of optional inputs, in this
c     case the maximum number of internally defined steps
c     allowed during one call to the solver.
c
c     usage: integral <xs> <xf> [<tol>]
c===========================================================
      program       integral

      implicit      none

      integer       iargc
      real*8        r8arg

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

c-----------------------------------------------------------
c     Command line arguments: integration limits and LSODA
c     (absolute) error tolerance---use a stringent default
c     tolerance.
c-----------------------------------------------------------
      real*8        xs,        xf,       tol

      real*8        default_tol
      parameter    ( default_tol = 1.0d-12 )

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

      integer       neq
      parameter    ( neq = 1 )

      real*8        y(neq)
```

```
      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     Note: Default value for 'mxstep' ('iwork(6)') is 500.
c-------------------------------------------------------------
      integer        mxstep
      parameter    ( mxstep = 50 000 )

      integer        i

c-------------------------------------------------------------
c     Parse command line arguments (initial values) ...
c-------------------------------------------------------------
      if( iargc() .lt. 2 ) go to 900

      xs   = r8arg(1,r8_never)
      if( xs .eq. r8_never ) go to 900
      xf   = r8arg(2,r8_never)
      if( xf .eq. r8_never ) go to 900
      tol  = r8arg(3,default_tol)

c-------------------------------------------------------------
c     Use pure absolute control.
c-------------------------------------------------------------
      itol  = 1
      rtol  = 0.0d0
      atol  = tol

      itask = 1
c-------------------------------------------------------------
c     Set the optional inputs as well as the flag which
c     tells LSODA optional inputs are being used.  A value
c     of 0 or 0.0d0 for any of the optional inputs tells
c     LSODA to use the internal default.
c-------------------------------------------------------------
      do i = 5 , 10
        iwork(i) = 0
        rwork(i) = 0.0d0
      end do
      iwork(6) = mxstep
      iopt  = 1

c-------------------------------------------------------------
c     Have LSODA compute the Jacobian numerically if
c     necessary (it won't be in this case!)
c-------------------------------------------------------------
      jt    = 2

c-------------------------------------------------------------
c     Initialize the integral.
c-------------------------------------------------------------
      y(1) = 0.0d0

c-------------------------------------------------------------
c     Integrate from x = xs to x = xf.  Note that LSODA
c     overwrites 'xs' with x-value in use at end of
c     integration (normally 'xf').
c-------------------------------------------------------------
      istate = 1

      call lsoda(fcn,neq,y,xs,xf,
     &           itol,rtol,atol,itask,
     &           istate,iopt,rwork,lrw,iwork,liw,jac,jt)

c-------------------------------------------------------------
c     Check return code, write result to standard output if
c     integration was successful, or message to standard
c     error otherwise.
c-------------------------------------------------------------
      if( istate .ge. 0 ) then
         write(*,*) y(1)
      else
         write(0,*) 'integral:  Error return ', istate,
     &              ' from LSODA'
      end if

c-------------------------------------------------------------
c     Normal exit.
c-------------------------------------------------------------
      stop

c-------------------------------------------------------------
c     Usage exit.
c-------------------------------------------------------------
 900  continue
         write(0,*) 'usage: integral <xs> <xf> [<tol>]'
      stop
      end
```

Source file: `fcn.f`

```
c=============================================================
c     Implements ODE for computation of definite integral of
c
c          exp(-x^2)
c=============================================================
      subroutine fcn(neq,x,y,yprime)
         implicit    none

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

         yprime(1) = exp(-x**2)

         return
      end

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

         return
      end
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = integral

all: $(EXECUTABLES)

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

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

Source file: `Output on lnx1`

```
##############################################################
# Building 'integral' and sample output on the lnx machines
##############################################################
lnx1% pwd; ls
/home/phys410/ode/integral
Makefile  fcn.f  integral.f

lnx1% make
pgf77 -g -c integral.f
pgf77 -g -c fcn.f
pgf77 -g -L/usr/local/PGI/lib integral.o \
               fcn.o -lp410f -lodepack \
               -llinpack -lblas -o integral

##############################################################
# Usage
##############################################################
lnx1% integral
 usage: integral <xs> <xf> [<tol>]

##############################################################
# We can check the results using the following Maple
# code (or similar)
#
# > Digits := 25;
# > evalf(int(exp(-x^2),x=0.0..5.0));
#
#       .8862 2692 5451 3954 7538 24606
##############################################################
lnx1% integral 0.0 5.0
   0.8862 2692 5451 8431
                    ^

##############################################################
# > evalf(int(exp(-x^2),x=0.0..100.0));
#
#       .8862 2692 5452 7580 1364 90837
##############################################################
lnx1% integral 0.0 100
   0.8862 2692 5447 2388
                 ^

##############################################################
# Repeat previous computation with less stringent tolerance,
# note that answer is (roughly) correspondingly less
# accurate.
##############################################################
lnx1% integral 0.0 100.0 1.0d-6
   0.8862 2897 2928 0249
           ^
```

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-------------------------------------------------------------
      istate = 1
      do while( t .le. tmax )
         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`

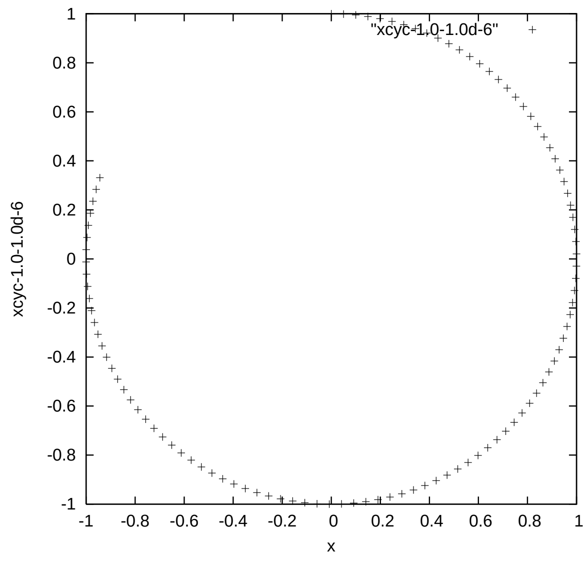


Figure file: `../twobody/dJtot-1.0-1.0d-6.ps`

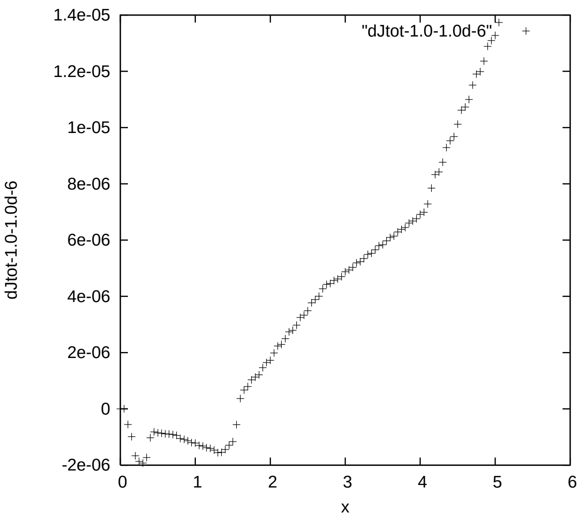


Figure file: `../twobody/dEtot-1.0-1.0d-6.ps`

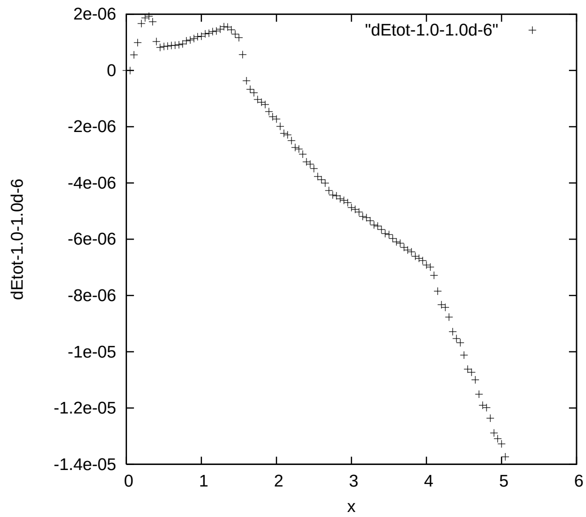


Figure file: `../twobody/xcyc-1.0-1.0d-10.ps`

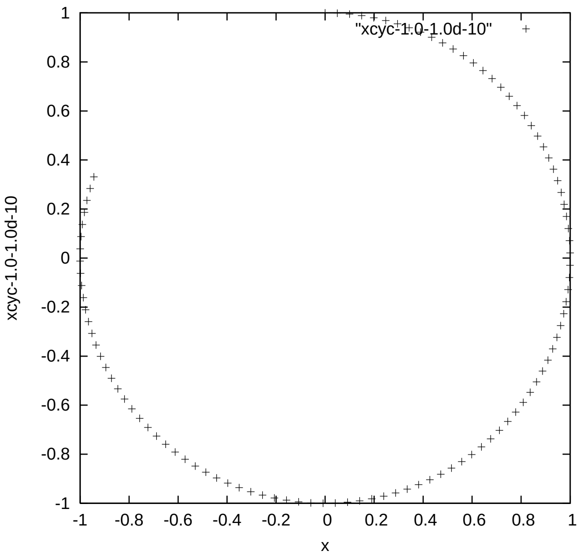


Figure file: `../twobody/dEtot-1.0-1.0d-10.ps`

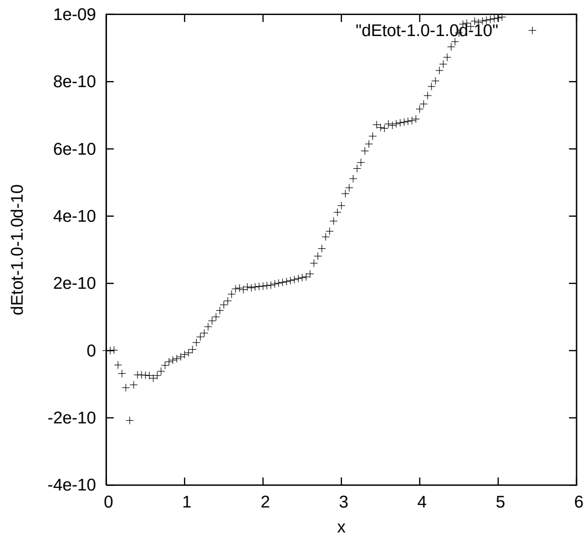


Figure file: `../twobody/xcyc-0.8-1.0d-10.ps`

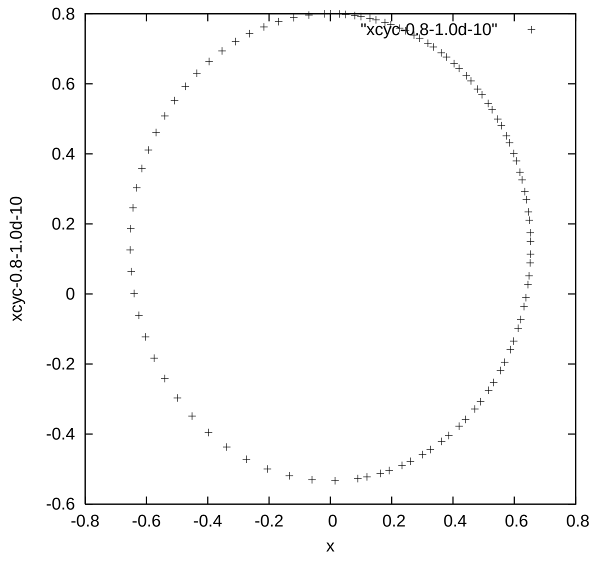


Figure file: `../twobody/dJtot-1.0-1.0d-10.ps`

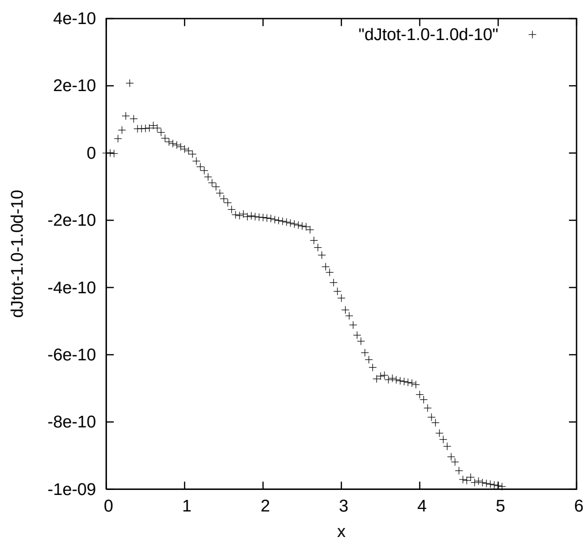


Figure file: `../twobody/dEtot-0.8-1.0d-10.ps`

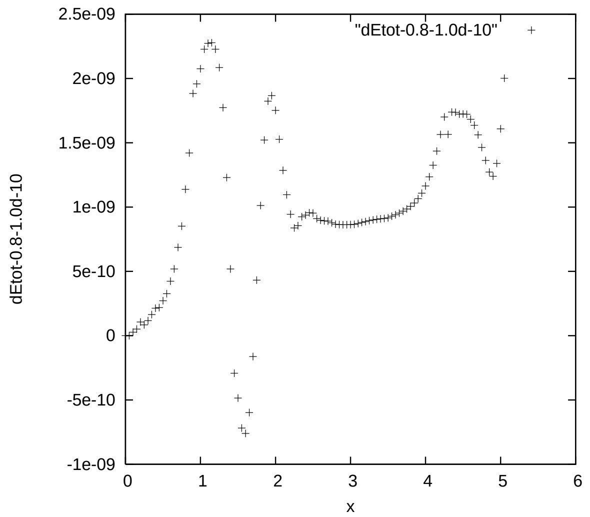
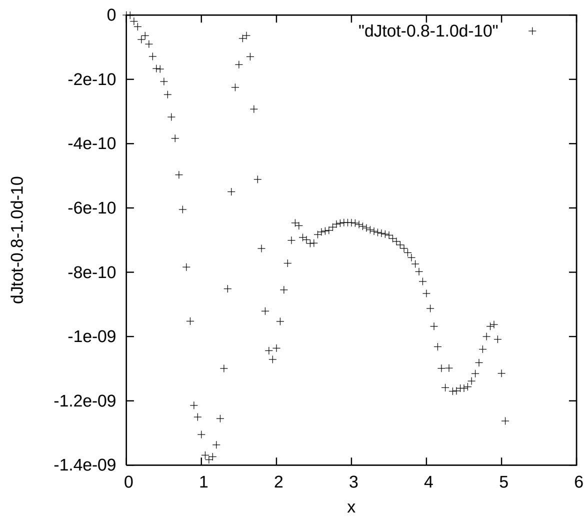


Figure file: `../twobody/dJtot-0.8-1.0d-10.ps`



Source file: `dumb.f`

```
c===========================================================
c     dumb: Uses LSODA to integrate equations of motion for
c     orbiting dumbbell.
c
c-----------------------------------------------------------
c     usage: dumb <y_0> <tmax> <dtout> <tol>
c
c            Specify <y_0> = 1.0 for circular orbit
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    = 2.0
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

      m1bym2 = 2.0d0
      mu     = 1.0d0 / (1.0d0 + m1bym2)
      d      = 0.3d0

      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-------------------------------------------------------------
      istate = 1
      do while( t .le. tmax )
         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`

```
#!/bin/sh -x

X="off"

test -f dumb || make

# High tolerance circular orbit
test -f circular || \
   dumb 1.0 1000.0 0.05 1.0d-12 > circular

if [ $X = "on" ]; then
   echo 2 2 1 > ppinput
   nf _1 _2 _3 0.0 _4 _5 0.0 < circular >> ppinput
   xfpp3d < ppinput
fi
# High tolerance elliptical orbit
test -f elliptical || \
   dumb 1.2 1000.0 0.05 1.0d-12 > elliptical

if [ $X = "on" ]; then
   echo 2 2 1 > ppinput
   nf _1 _2 _3 0.0 _4 _5 0.0 < elliptical >> ppinput
   xfpp3d < ppinput
fi

# Low tolerance circular orbit
test -f elliptical-lo || \
    dumb 1.2 1000.0 0.05 1.0d-6 > elliptical-lo
if [ $X = "on" ]; then
   echo 2 2 1 > ppinput
   nf _1 _2 _3 0.0 _4 _5 0.0 < elliptical-lo >> ppinput
   xfpp3d < ppinput
fi

test -f ppinput && /bin/rm ppinput

# Make plots

# Column:    1  2  3  4  5 6     7     8    9    10   11
# Quantity:  t x1 y1 x2 y2 theta omega KE_t KE_r PE_g E_tot

test -f ket-e || nth 1 8 < elliptical > ket-e
test -f ker-e || nth 1 9 < elliptical > ker-e
test -f peg-e || nth 1 10 < elliptical > peg-e
test -f etot-e || nth 1 11 < elliptical > etot-e

test -f ket-el || nth 1 8 < elliptical-lo > ket-el
test -f ker-el || nth 1 9 < elliptical-lo > ker-el
test -f peg-el || nth 1 10 < elliptical-lo > peg-el
test -f etot-el || nth 1 11 < elliptical-lo > etot-el

test -f om-c.ps || gnuplot<<END
set terminal postscript portrait
set output "om-c.ps"
set size square
set title "Orbiting Dumbbell Problem\nCircular Orbit\
 -- Tolerance=10(-12)"
set xlabel "t"
set ylabel "omega"
plot [0:60] [0:2.2] "circular" using (\$1):(\$7)\
 notitle with lines
END


test -f om-e.ps || gnuplot<<END
set terminal postscript portrait
set output "om-e.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit\
 -- Tolerance=10(-12)"
set xlabel "t"
set ylabel "omega"
plot [0:1000] [-1:3.5] "elliptical" using (\$1):(\$7)\
  notitle with lines
END


test -f om-ez.ps || gnuplot<<END
set terminal postscript portrait
```

```
set output "om-ez.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit\
 -- Tolerance=10(-12)"
set xlabel "t"
set ylabel "omega"
plot [0:200] [-0.5:2.8] "elliptical" using (\$1):(\$7)\
  notitle with lines
END


test -f kerot-12.ps || gnuplot<<END
set terminal postscript portrait
set output "kerot-12.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit\n\
Rotational Kinetic Energy -- Tolerance=10(-12)"
set xlabel "t"
set ylabel " "
plot [0:1000] [0:0.1250] "elliptical" using (\$1):(\$9)\
  notitle with lines
END

test -f e-12.ps || gnuplot<<END
set terminal postscript portrait
set output "e-12.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit -- \
Energy uantities -- Tolerance=10(-12)\nTop to Bottom: \
KE_t, KE_r, E_tot, PE_g"
set xlabel "t"
set ylabel " "
plot [0:1000] [-2:1.5] \
     "ket-e" notitle with lines, \
     "ker-e" notitle with lines, \
     "peg-e" notitle with lines, \
     "etot-e" notitle with lines
quit
END

etot0=`lino 1 < etot-e | nth 2`
test -f etot-12.ps || gnuplot<<END
set terminal postscript portrait
set output "etot-12.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit -- \
Deviation in Total Energy -- Tolerance=10(-12)"
set xlabel "t"
set ylabel " "
plot [0:1000] "etot-e" using (\$1):($etot0-\$2) notitle with lines
quit
END

test -f e-6.ps || gnuplot<<END
set terminal postscript portrait
set output "e-6.ps"
set size square
set title "Orbiting Dumbbell Problem\nElliptical Orbit -- \
Energy Quantities -- Tolerance=10(-6)\nTop to Bottom: \
KE_t, KE_r, E_tot, PE_g"
set xlabel "t"
set ylabel " "
plot [0:1000] [-2:1.5] \
     "ket-el" notitle with lines, \
     "ker-el" notitle with lines, \
     "peg-el" notitle with lines, \
     "etot-el" notitle with lines
quit
END

ls -lt *ps
```

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

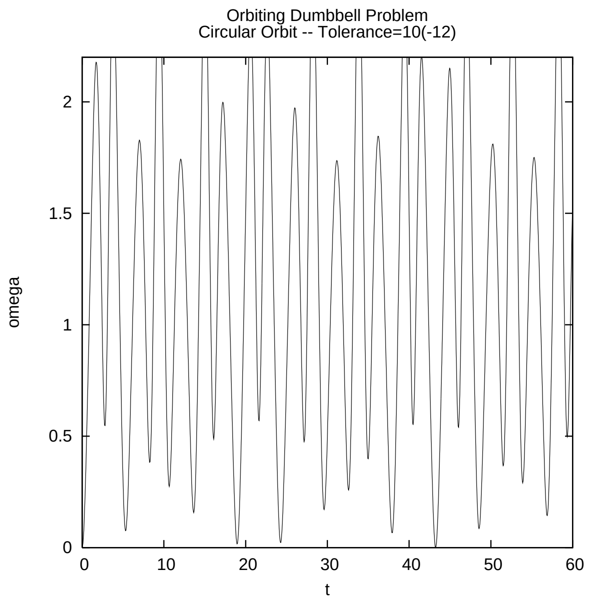


Figure file: ../dumb/om-e.ps

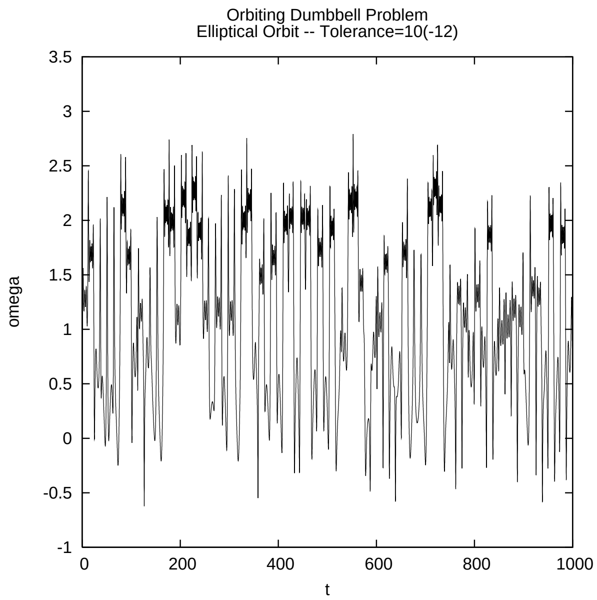


Figure file: `../dumb/om-ez.ps`

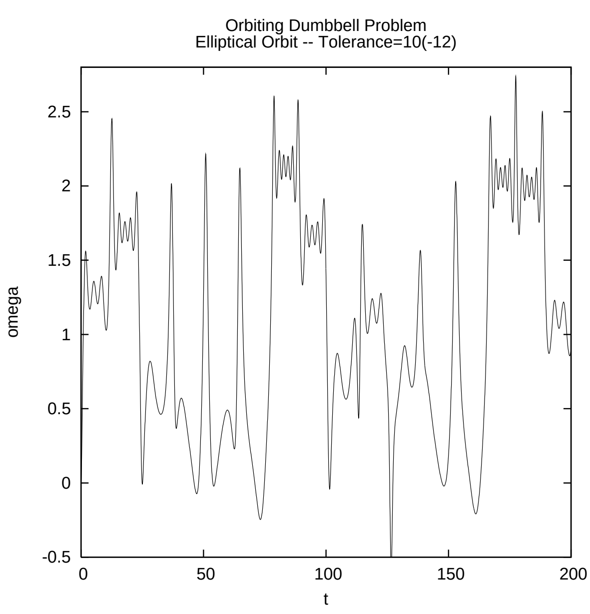


Figure file: `../dumb/e-12.ps`

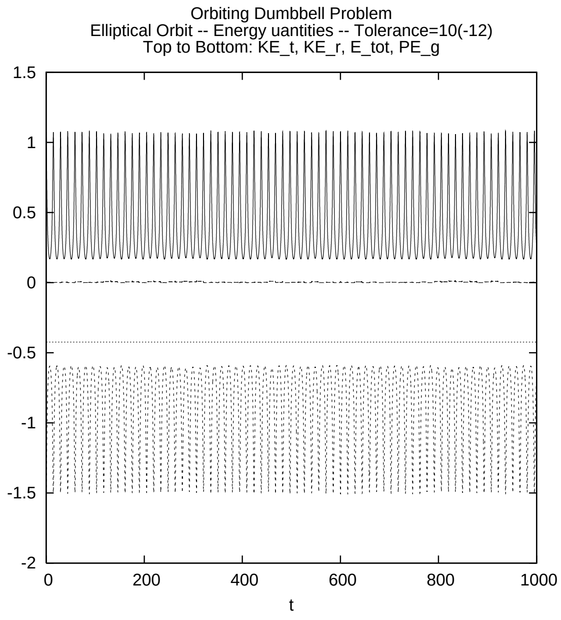


Figure file: `../dumb/e-6.ps`

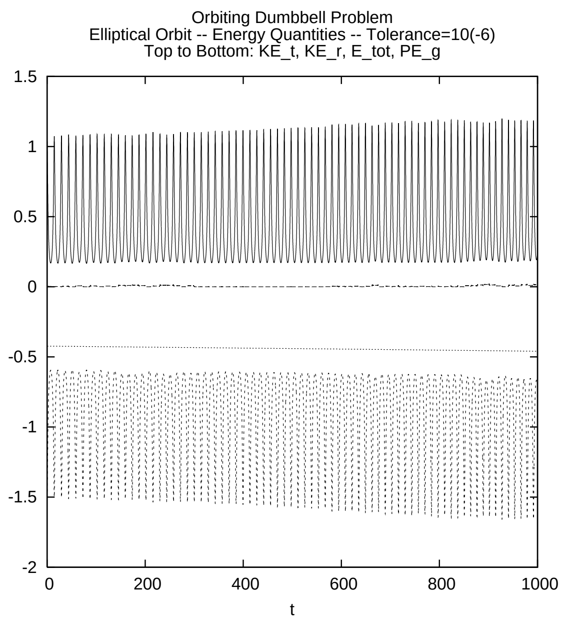
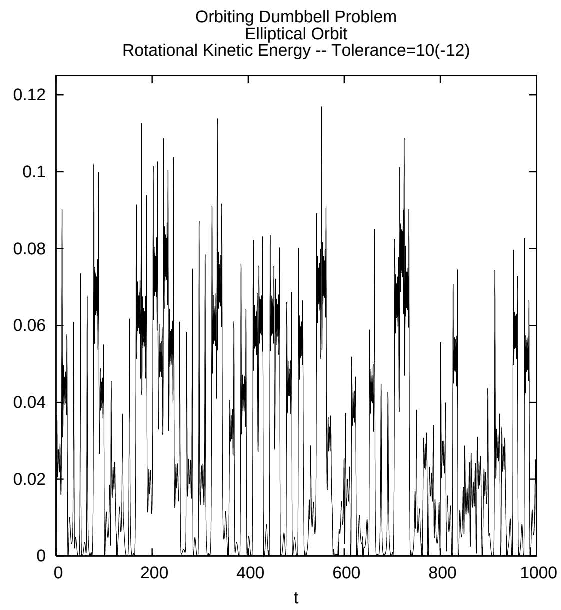


Figure file: `../dumb/kerot-12.ps`



Source file: `wave.f`

```fortran
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     usage: wave <xlevel> <olevel> <tfinal> <dtout> <tol>
c
c        <xlevel>  := Defines spatial discretization;
c                     spatial grid has 2**xlevel + 1 pts.
c        <olevel>  := Controls number of spatial values that
c                     are output to standard out for plotting
c                     via gnuplot; every 2**(xlevel - olevel)
c                     value is output.
c        <tfinal>  := Final integration time.
c        <dtout>   := Output time interval.
c        <tol>     := LSODA tolerance.
c-----------------------------------------------------------
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 'xvs' interface to generate
c     .sdf files which can subsequently be visualized using
c     'xvs' visualization server.  See links in course
c     Software 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
      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)    = xmax
      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 xvs('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-----------------------------------------------------------
      istate = 1
      do while( t .lt. tfinal )
         tout = t + dtout
c-----------------------------------------------------------
c        Call lsoda to integrate system on [t ... tout]
```

```
c-----------------------------------------------------------
         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 xvs('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`

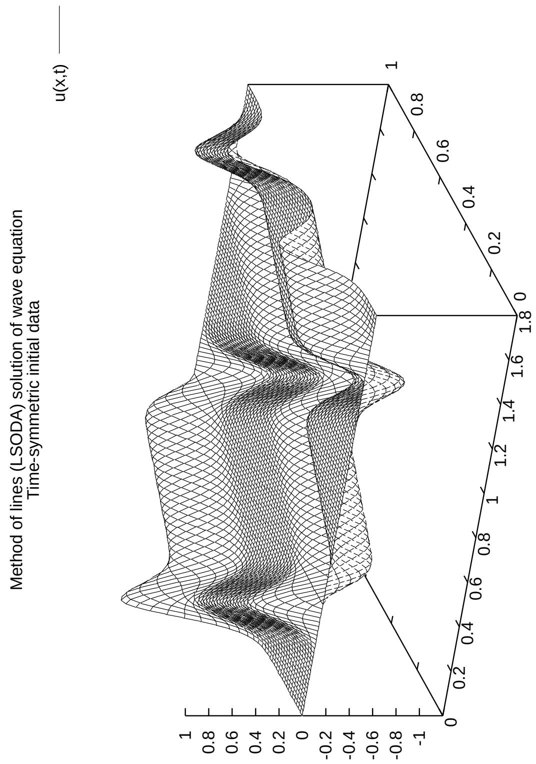
```
.IGNORE:

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`

```
#!/bin/sh -x
#-----------------------------------------------------------
# Wave: script which runs 'wave' and then graphs output
# using gnuplot
#-----------------------------------------------------------
test -f wave || make

# Set the command-line parameters for wave
xlevel=8
olevel=6
tfinal=1.8
dtout=0.02
tol=1.0e-5

# Warn the user re the run time
echo "This may take a couple of minutes or so.  Please be patient."

# Generate the solution
wave $xlevel $olevel $tfinal $dtout $tol > out8

# Plot the solution
gnuplot<<END
set terminal postscript portrait
set title "Method of lines (LSODA) solution of wave \
equation\nTime-symmetric initial data"
set output "out8.ps"
set ticslevel 0.1
set parametric
set hidden
splot "out8" title "u(x,t)" with lines
quit
END
ls -lt *ps
```

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-----------------------------------------------------------
      istate = 1
      do while( x .lt. xmax )
         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) * u
         else
            yprime(2) = -E * u
         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.  Observe
# that due to the previously noted facts, will generally need
# Elo > Ehi.  Also note that E(x0) < 0 (bound states), and
# at least for a certain range of x0 (e.g. 2.0 <= x0 <= 6.0),
# a suitable initial bracket is [0.0,-1.0]
#
# Output accumulated in directories/files
#
#   x0=<x0>/E=<E>
############################################################

P=`basename $0`

usage() {
printf "$P <x0> <Elo> <Ehi> <Etol> <xmax> <dxout>"
printf " <lsoda tol> [<xvstrace>]\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=$7; xvstrace=${8-false};
   case $xvstrace in
   true|false) ;;
   *) "xvstrace must be 'true' or 'false'";;
   esac;;
*) usage;;
esac

# Set tolerance for binary search
export BSTOL=$Etol

# Make executable if necessary
test -f deut || make deut

# 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
   $xvstrace && nth 1 2 < $ofile | xvn $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

# Save results of final integration  ...
nth 1 2 < $ofile > $dir/solution

# ... and print summary to 'deut-results'
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=2.0\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=2.0\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`

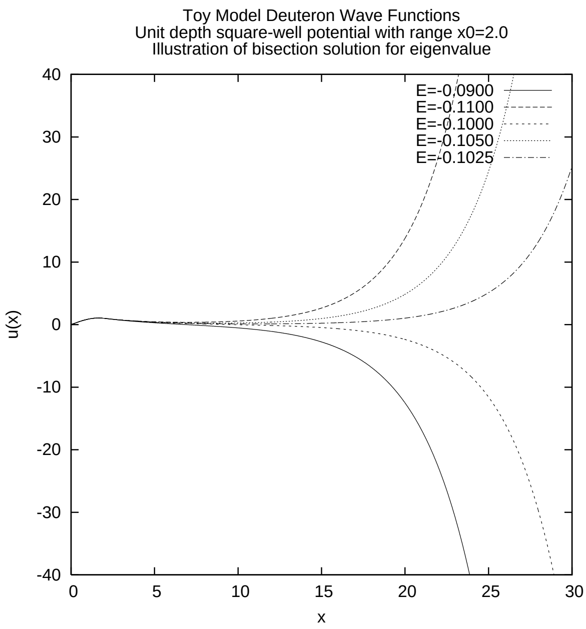


Figure file: `../deut/zshoot.ps`

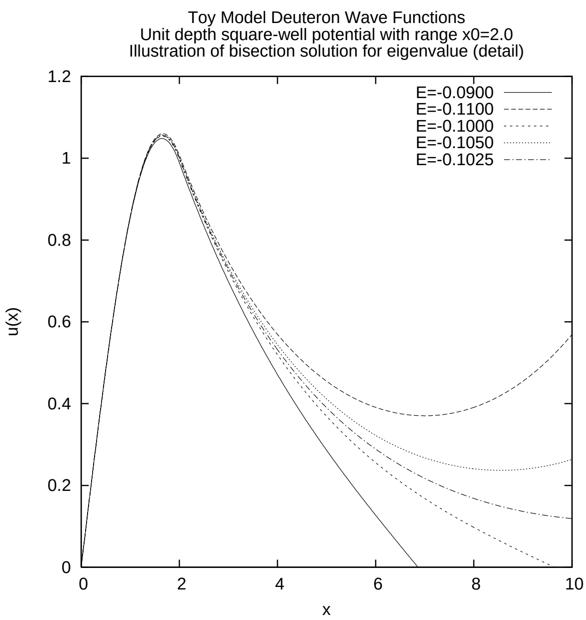


Figure file: `../deut/u.ps`