```
c===========================================================
c     Program demonstrating use of 'lsoda' to evaluate
c     a definite integral.
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         xlim

      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     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     Set LSODA parameters ... use pure absolute control.
c------------------------------------------------------------
      itol  = 1

      rtol  = 0.0d0
      atol  = tol

      itask = 1
      iopt  = 0
      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
```

```
c-----------------------------------------------------------------------
c     Computes definite integral of
c
c          exp(-x^2)
c
c-----------------------------------------------------------------------

c-----------------------------------------------------------------------
c     Implements differential equations.
c     Called by ODEPACK routine LSODA.
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     Implements Jacobian  (optional) ...
c-----------------------------------------------------------------------
      subroutine jac
         implicit    none

         return
      end
```

```
#############################################################
# Building 'integral' and sample output on the SGIs
#############################################################
sgi1% pwd; ls
/usr2/people/phy329/ode/ex2/integral
Makefile     fcn.f        integral.f

sgi1% make
f77 -g -64 -c integral.f
f77 -g -64 -c fcn.f
f77 -g -64 -L/usr/local/lib integral.o fcn.o -lp329f -lodepack \
               -llinpack -lblas -o integral

#############################################################
# Usage
#############################################################
sgi1% 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));
#
#       .8862269254513954753824605
#############################################################
sgi1% integral 0.0 5.0
   0.8862269254350943
               ^
```

```
##############################################################
# > evalf(int(exp(-x^2),x=0.0..10.0));
#
#       .8862269254527580136490835
##############################################################
sgi1% integral 0.0 10.0
   0.8862269254428063
               ^

##############################################################
# > evalf(int(exp(-x^2),x=0.0..100.0));
#
#       .8862269254527580136490835
##############################################################
sgi1% integral 0.0 100.0
   0.8862269254522580
                 ^

##############################################################
# Repeat previous computation with less stringent tolerance,
# note that answer is (roughly) correspondingly less
# accurate.
##############################################################
sgi1% integral 0.0 100.0 1.0d-6
   0.8862289729280245
          ^

##############################################################
# Beware of strange results which can occur if integrand
# 'underflows'
##############################################################
sgi1% integral -100.0 -100.0
   0.0000000000000000E+00
```

```
c===========================================================
c     2body:  Integrates restricted gravitational 2-body
c     problem using ODEPACK LSODA routine.
c
c     usage: 2body <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         tol
      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,*) '2body:  Error return ', istate,
     &                 ' from LSODA '
            write(0,*) '2body: 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: 2body <x0> <y0> <vx0> <vy0> '//
     &              '<tmax> <dt> [<tol>]'
      stop
      end
```

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

         include    'fcn.inc'

         return
      end
```

```
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
```

```
#! /bin/sh

############################################################
# This shell script is a "front-end" to 2body 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.
#
# The script is also intended as a general illustration of
# shell programming.  A key idea here is that many things
# which are cumbersome in Fortran (or C, or C++)
# (such as file manipulation, and piecing together different
# commands) are easily accomplished with a script.
#
# See the Unix Programming Environment by Kernighan and Pike
# for classic documentation on shell programming and more!
############################################################

############################################################
# Note that the first line of this file is very important.
# It allows Unix to identify the contents of the file as
# a shell script ('/bin/sh' is the shell executable on
# ALL non-brain-dead Unix systems), so that, provided the
# file is made executable and resides in a directory in
# the current path, typing the name of the file will
# result in execution of the script.  The space between
# the '#!' and '/bin/sh'is recommended for maximum script
# portability.
#
# A variant initial line, particularly useful for script
# development is
#
# #! /bin/sh -x
```

```
#
# The '-x' flag enables tracing, so that assignments of
# shell variables, execution of commands etc. will be
# echoed to standard error.
############################################################

############################################################
# ASSIGNMENT STATEMENTS
#
# The following code assigns some default values to some
# shell variables.
#
# NOTE: There can be *no* white
# space around the '=' in an assignment statement. If,
# for example you type
#
# tmax = 5.0
#
# then, when you execute the script, you will get the
# cryptic error message
#
# tmax: Not found
#
# since the shell will be interpreting the line as the
# *command* 'tmax' with arguments '= 5.0'.
############################################################
tmax=5.0
dt=0.05
tol=1.0d-6
```

```
############################################################
# FUNCTIONS
#
# Usage function (subroutine).  Illustrates one way to
# "do usage" in shell script (using 'cat' plus a
# "here document"), as well as the useful "here document"
# construction.
#
# Here documents can be used anywhere
# in a shell script (and in C-shell, Korn-shell, perl etc.
# script) to provide "in-place" input for the standard input
# of a command.  You can refer to the man page on 'sh'
# for full details, but the idea and mechanics are
# simple.  To provide "in-place" input to an arbitrary
# command, append '<<END' after the command name and
# all arguments.  Be certain that there is no white
# space between the '<<' and 'END' or after 'END'.
# Subsequent lines are then the standard input to
# the command.  A line which *exactly* matches
# the string 'END' (i.e. 'grep ^END$' succeeds)
# signals end-of-file (so be sure you have such a line
# in your script!).  Again, beware of leading or trailing
# white space in the end-of-file marker.  Finally, note that
# the string'END' is arbitrary---you can use essentially any
# string you wish (but be sure to use the identical string in
# both contexts)---'END' is simply my convention.
#
# Here documents wouldn't be half as interesting if
# it weren't for the fact that they are partially interpreted
# by the shell.  In particular, shell variables are evaluated
# as are `command [arguments]` constructs (recall that
# we can view such constructs as "active expressions" of
# their standard outputs).  Thus, when the Usage subroutine
# executes `date` will be replaced by the output from the
```

```
# 'date' command and  $tol will be replaced with the
# specific value assigned above.
#############################################################
Usage() {
cat<<END
`date`
usage: Doit <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 with non-zero status (failure).
#############################################################
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.   Again, note use of "here document";
# indentation of "here document" is strictly for readability.
############################################################
gnuplot_it(){
gnuplot<<END
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "$1"
   set output "$1.ps"
   plot "$1"
   quit
END
```

```
############################################################
# IF STATEMENTS
#
# The following is shell-mnemonic for "if the second
# argument exists".  Note that the shell 'if' statement
# is of the form
#
# if list then list [ elif list then list ] ... [ else list ] fi
#
# where a 'list' is a sequence of pipelines separated by ;,
# &, # &&, or ||, and optionally terminated by ; or &.
# 'list' is executed, and if it returns a zero exit
# status (normal return), the 'list' following 'then'
# is executed, otherwise the 'list' following 'elif'
# is executed---if *it* returns a zero exit status, the
# 'list' following the next 'then' is executed etc.
#
# A particularly useful command to use with a shell 'if'
# is 'test', which has usage
#
# test expr
#
# or
#
# [ expr ]
#
# Note that '[' and ']' really ARE commands on a Unix system.
# (Try 'which \['). Both forms of the command evaluate
# 'expr' (which can contain various logical operators
# and tests---see 'man test' for more information) as
# a Boolean, returning 0 if the value is true and non-zero
# otherwise.
############################################################
```

```
if test "${2}undefined" != undefined; then
############################################################
# Illustrates alternate (and less portable, so it is
# claimed) form of 'test'
############################################################
   if [ -f $1.ps ]; then
      (gv $1.ps) &
   else
       echo "gnuplot_it: $f.ps does not exist"
   fi
fi
}

############################################################
# ARGUMENT HANDLING and CASE STATEMENT
#
# '$#' is a special shell variable which evaluates to
# the number of arguments supplied to the script.  Each
# clause in a case statment is of the form
#
# pattern [ | pattern ]...)  list;;
#
# where each pattern is used for 'grep-style' pattern
# matching against the case variable.  In this case
# we simply match against valid numbers of arguments
# (1, 2 or 3), but note that the pattern-matching basis
# of the case statement makes it a particularly powerful
# and useful construct. Commands may be separated by
# semicolons and/or new-lines. '*)' is a mnemonic for
# 'default clause' and must come LAST in the case
# enumeration, since matching is done clause by
# clause until a match is found.
#
```

```
# Note that arguments to the script are available in
# the special variables '1, 2, 3 ...'---i.e.
#
# y0=$1
#
# assigns the first argument supplied to the script to
# the shell variable 'y0' (The special vbl.
# '0' stores the name of the script).  Also note
# the mnemonic typified by
#
# tol=${2-$tol}
#
# which reads "assign the second script argument to tol,
# with a default value of $tol should the argument not
# exist".
############################################################
case $# in
1|2) y0=$1; tol=${2-$tol};;
*) Usage;;
esac
############################################################
# Make sure the program has been compiled.
############################################################
make -f Makefile 2body
############################################################
# Create a 'tag' which identifies the computation (encodes
# 'y0' and 'tol'), and use it to create the name of the
# file to which '2body' output will be directed.  Note
# that the script does NOT completely clean up after itself
# in the sense that the output file 'oute_...' is not
# removed.
############################################################
tag="$y0"_"$tol"
ofile=oute_"$tag"
```

```
############################################################
# Run the program and save the output in $ofile.
############################################################
2body 0.0 $y0 1.0 0.0 $tmax $dt $tol > $ofile

############################################################
# Use 'nth' to create two column data files for as follows
#
#       xcyc_$tag:  xc   yc
#       xc_$tag:    t    xc
#       yc_$tag:    t    yc
#       dEtot_$tag  t    dEtot
#       dJtot_$tag  t    JEtot
############################################################
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 LOOPS
#
# General form:
#
# for name [ in word ... ] do list done
#
# In the case below, the shell variable 'f' takes on,
# successively, the values, xcyc_$tag, xc_$tag, ...
#
# Note the semi-colon between the iteration values and the
# 'do' keyword (a new-line will also work), and that 'done'
# terminates the loop.
############################################################
for f in xcyc_$tag xc_$tag yc_$tag dEtot_$tag dJtot_$tag; do
############################################################
#  Shell scripts inherit all environment variables, such
#  as 'GV'.  If 'GV' has not been set in
#  the invoking environment, '$GV' will evaluate to
#  the null string.
############################################################
   gnuplot_it $f $GV
   /bin/rm $f
done
/bin/ls -l *$tag*.ps

############################################################
# Normal exit (success)
############################################################
exit 0
```

```
#! /bin/sh

############################################################
# This shell script is a "front-end" to 2body 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.
#
# History: Doit
#
# This is Doit with the vast majority of the comments
# removed.
############################################################

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

############################################################
# Usage
############################################################
Usage() {
cat<<END
`date`
usage: Doit_s <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 0.760,1.0
   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
}
```

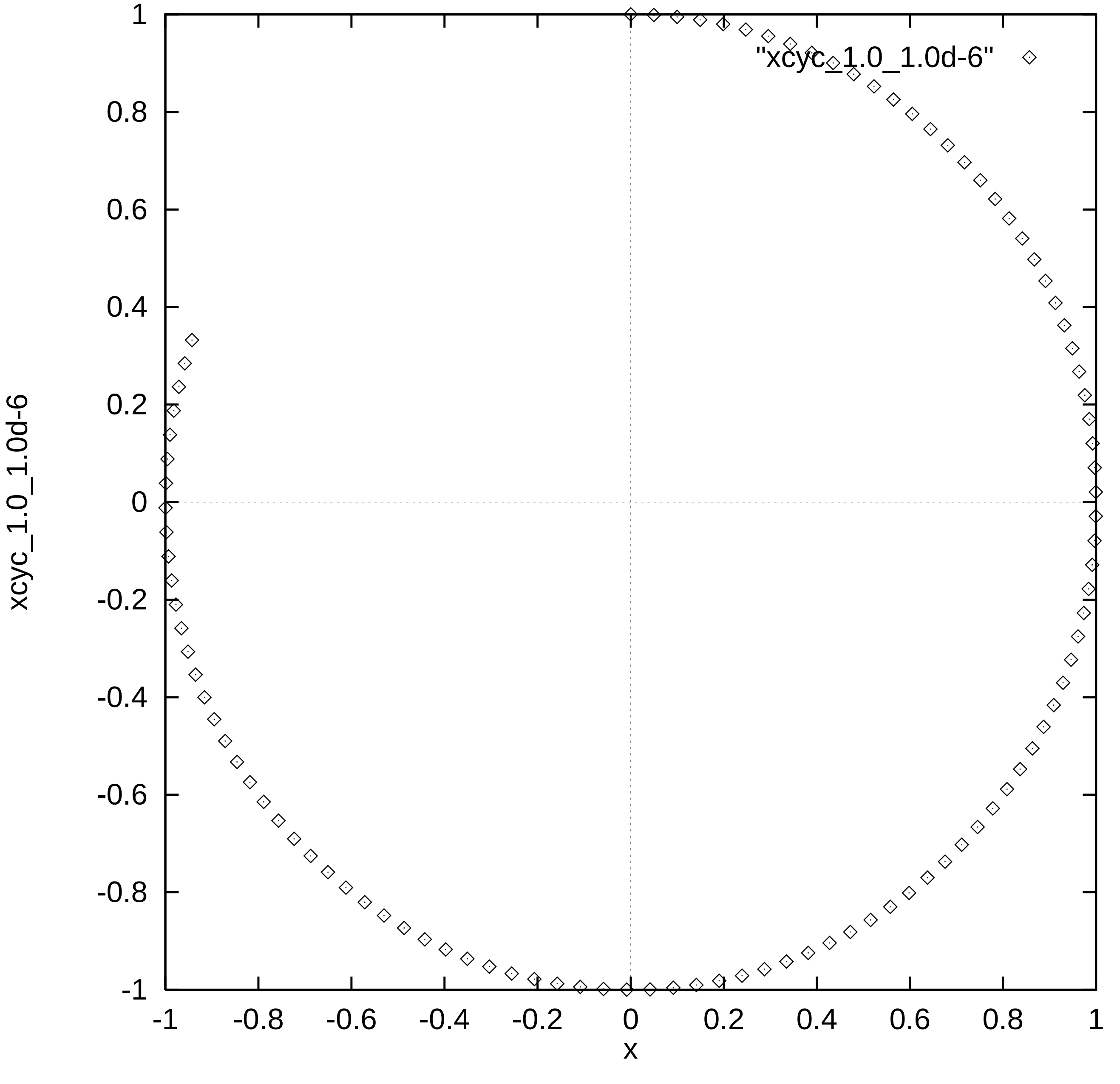
```
############################################################
# Argument handling
############################################################
case $# in
1|2) y0=$1; tol=${2-$tol};;
*) Usage;;
esac

############################################################
# Build application, run it, and process the results.
############################################################
make -f Makefile 2body

tag="$y0"_"$tol"
ofile=oute_$tag

2body 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
```

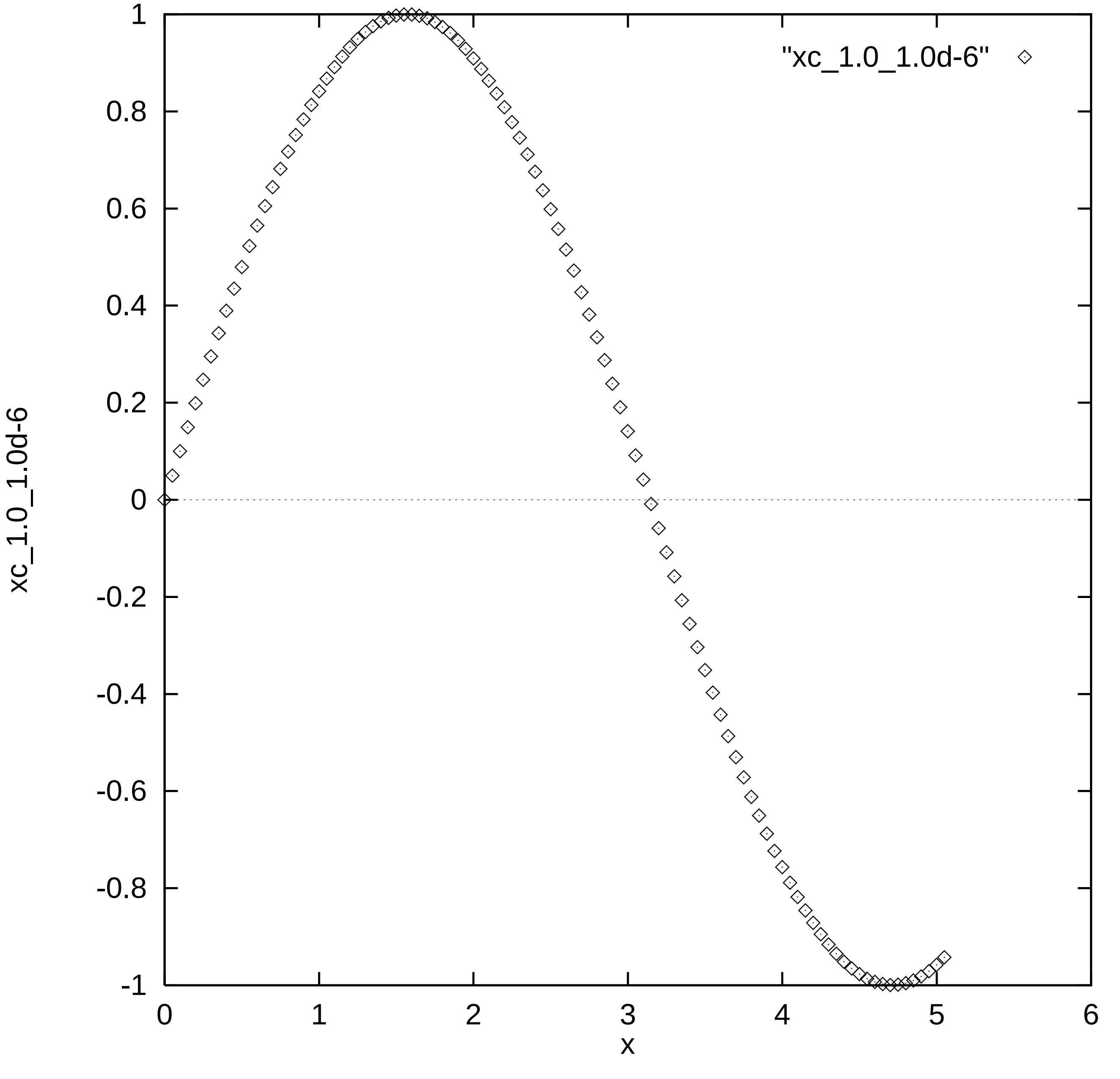
Output from 'Doit 1.0 1.0d-6' with tracing enabled

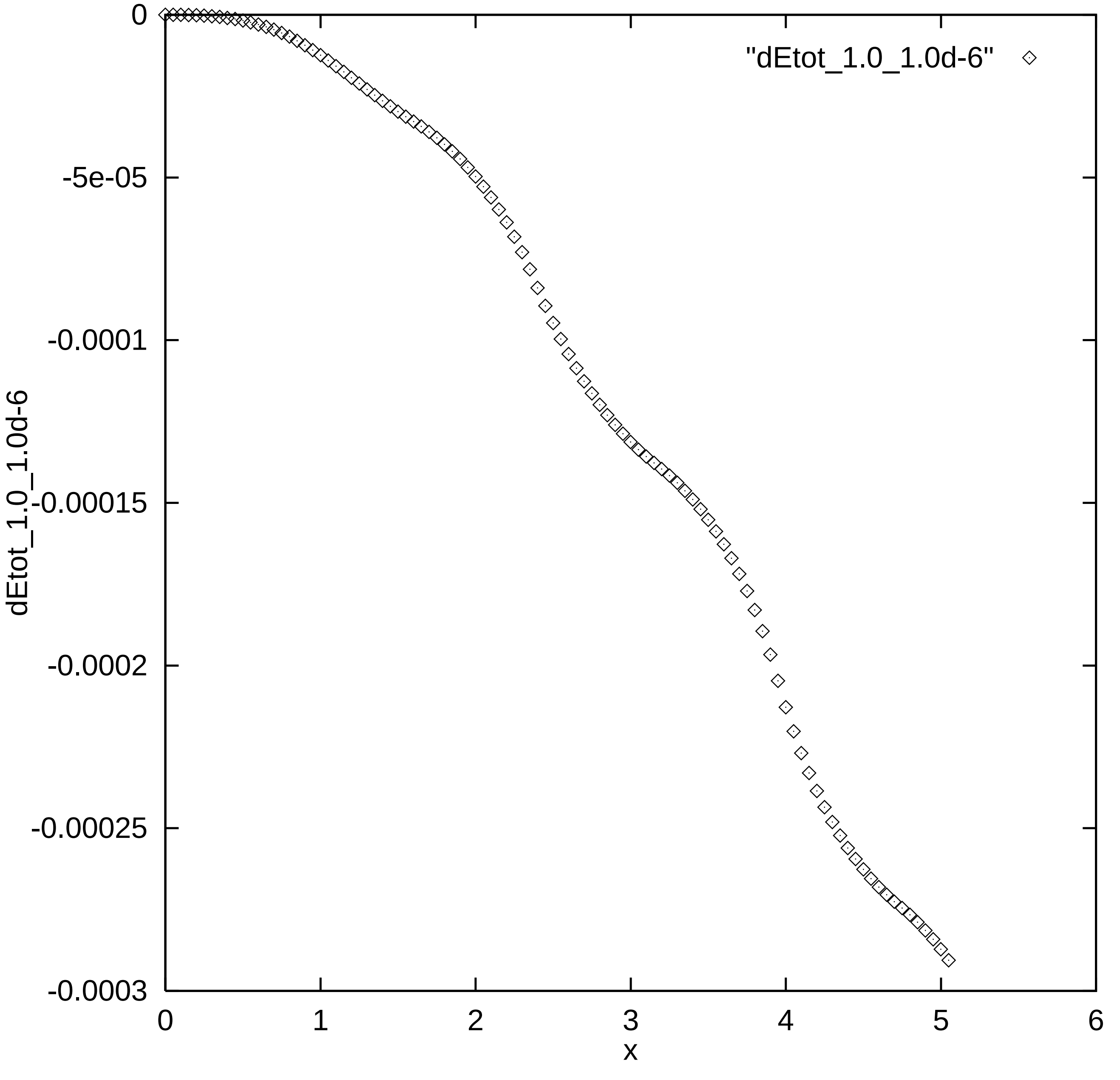
```
tmax=5.0
dt=0.05
tol=1.0d-6
y0=1.0
tol=1.0d-6
+ make -f Makefile 2body
f77 -g2 -mips4 -64 -c 2body.f
f77 -g2 -mips4 -64 -c fcn.f
f77 -g2 -mips4 -64 -L/usr/local/lib 2body.o fcn.o -lvs -lutilio -lvutil -lodepack -llinpack -lblas -o 2body
ld64: WARNING 84: /usr/local/lib/libvs.a is not used for resolving any symbol.
ld64: WARNING 84: /usr/local/lib/libvutil.a is not used for resolving any symbol.
tag=1.0_1.0d-6
ofile=oute_1.0_1.0d-6
+ 2body 0.0 1.0 1.0 0.0 5.0 0.05 1.0d-6
+ nth 2 3
+ nth 1 2
+ nth 1 3
+ nth 1 4
+ nth 1 5
+ gnuplot_it xcyc_1.0_1.0d-6
+ gnuplot
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "xcyc_1.0_1.0d-6"
   set output "xcyc_1.0_1.0d-6.ps"
   plot "xcyc_1.0_1.0d-6"
   quit
+ test undefined != undefined
+ /bin/rm xcyc_1.0_1.0d-6
```

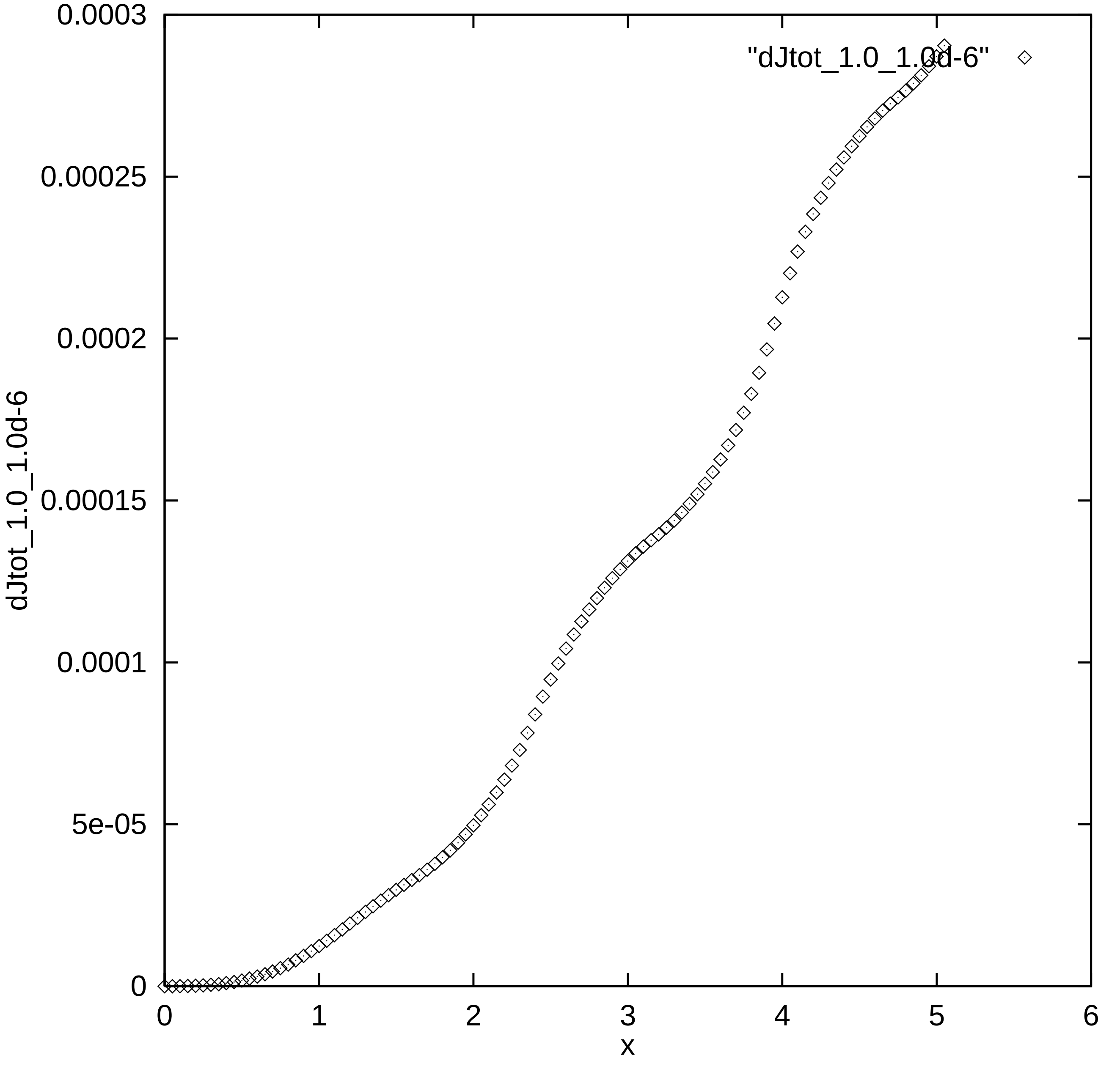
```
+ gnuplot_it xc_1.0_1.0d-6
+ gnuplot
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "xc_1.0_1.0d-6"
   set output "xc_1.0_1.0d-6.ps"
   plot "xc_1.0_1.0d-6"
   quit
+ test undefined != undefined
+ /bin/rm xc_1.0_1.0d-6
+ gnuplot_it yc_1.0_1.0d-6
+ gnuplot
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "yc_1.0_1.0d-6"
   set output "yc_1.0_1.0d-6.ps"
   plot "yc_1.0_1.0d-6"
   quit
+ test undefined != undefined
+ /bin/rm yc_1.0_1.0d-6
+ gnuplot_it dEtot_1.0_1.0d-6
+ gnuplot
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "dEtot_1.0_1.0d-6"
   set output "dEtot_1.0_1.0d-6.ps"
   plot "dEtot_1.0_1.0d-6"
   quit
+ test undefined != undefined
+ /bin/rm dEtot_1.0_1.0d-6
+ gnuplot_it dJtot_1.0_1.0d-6
```

```
+ gnuplot
   set terminal postscript portrait
   set size 0.760,1.0
   set xlabel "x"
   set ylabel "dJtot_1.0_1.0d-6"
   set output "dJtot_1.0_1.0d-6.ps"
   plot "dJtot_1.0_1.0d-6"
   quit
+ test undefined != undefined
+ /bin/rm dJtot_1.0_1.0d-6
+ /bin/ls -l dEtot_1.0_1.0d-6.ps dJtot_1.0_1.0d-6.ps xc_1.0_1.0d-6.ps xcyc_1.0_1.0d-6.ps yc_1.0_1.0d-6.ps
-rw-r--r--    1 matt     choptuik    4580 Nov 19 14:08 dEtot_1.0_1.0d-6.ps
-rw-r--r--    1 matt     choptuik    4568 Nov 19 14:08 dJtot_1.0_1.0d-6.ps
-rw-r--r--    1 matt     choptuik    4787 Nov 19 14:08 xc_1.0_1.0d-6.ps
-rw-r--r--    1 matt     choptuik    5015 Nov 19 14:08 xcyc_1.0_1.0d-6.ps
-rw-r--r--    1 matt     choptuik    4779 Nov 19 14:08 yc_1.0_1.0d-6.ps
+ exit 0
```

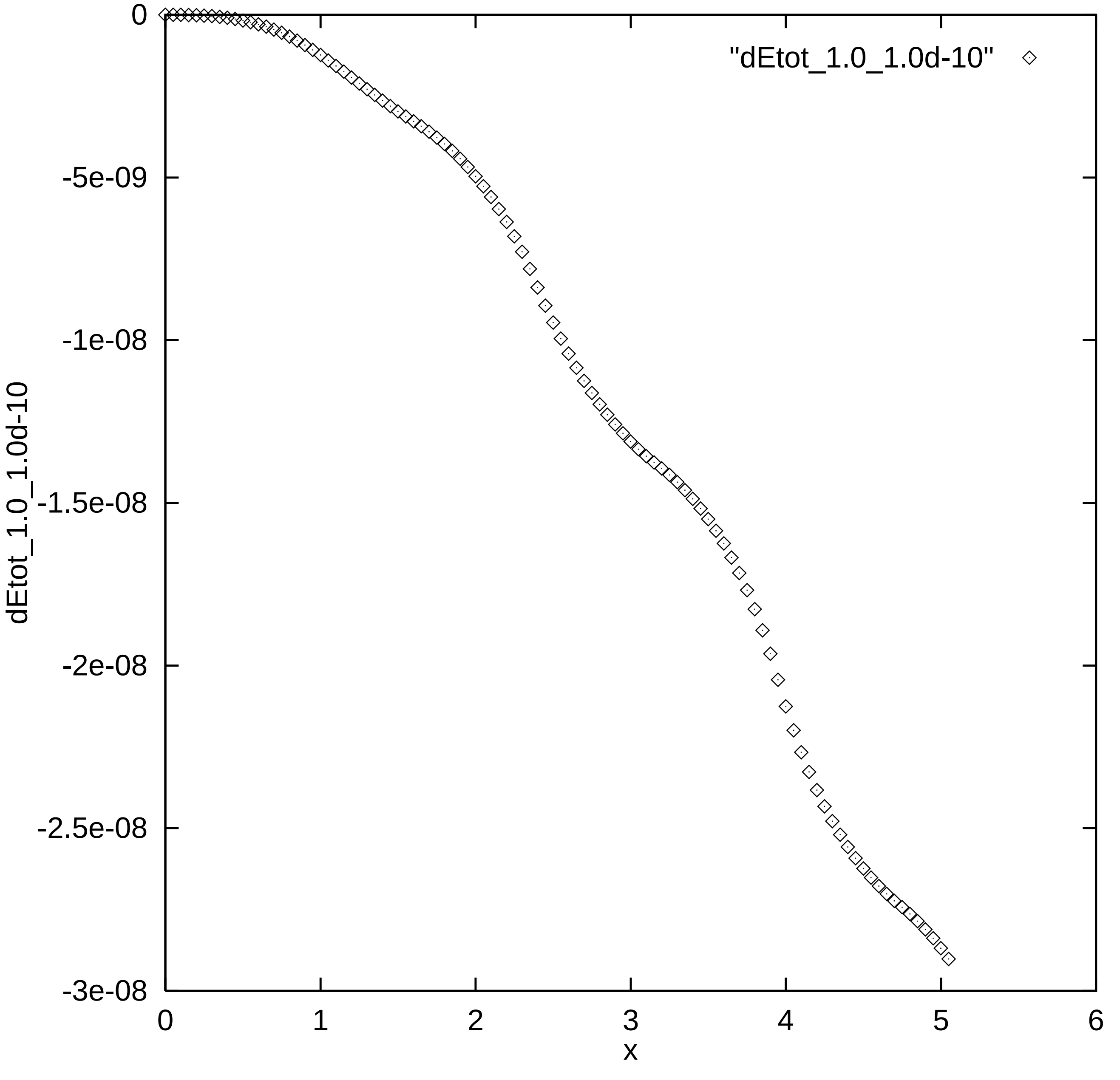

"xcyc_1.0_1.0d-6"
xcyc_1.0_1.0d-6
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

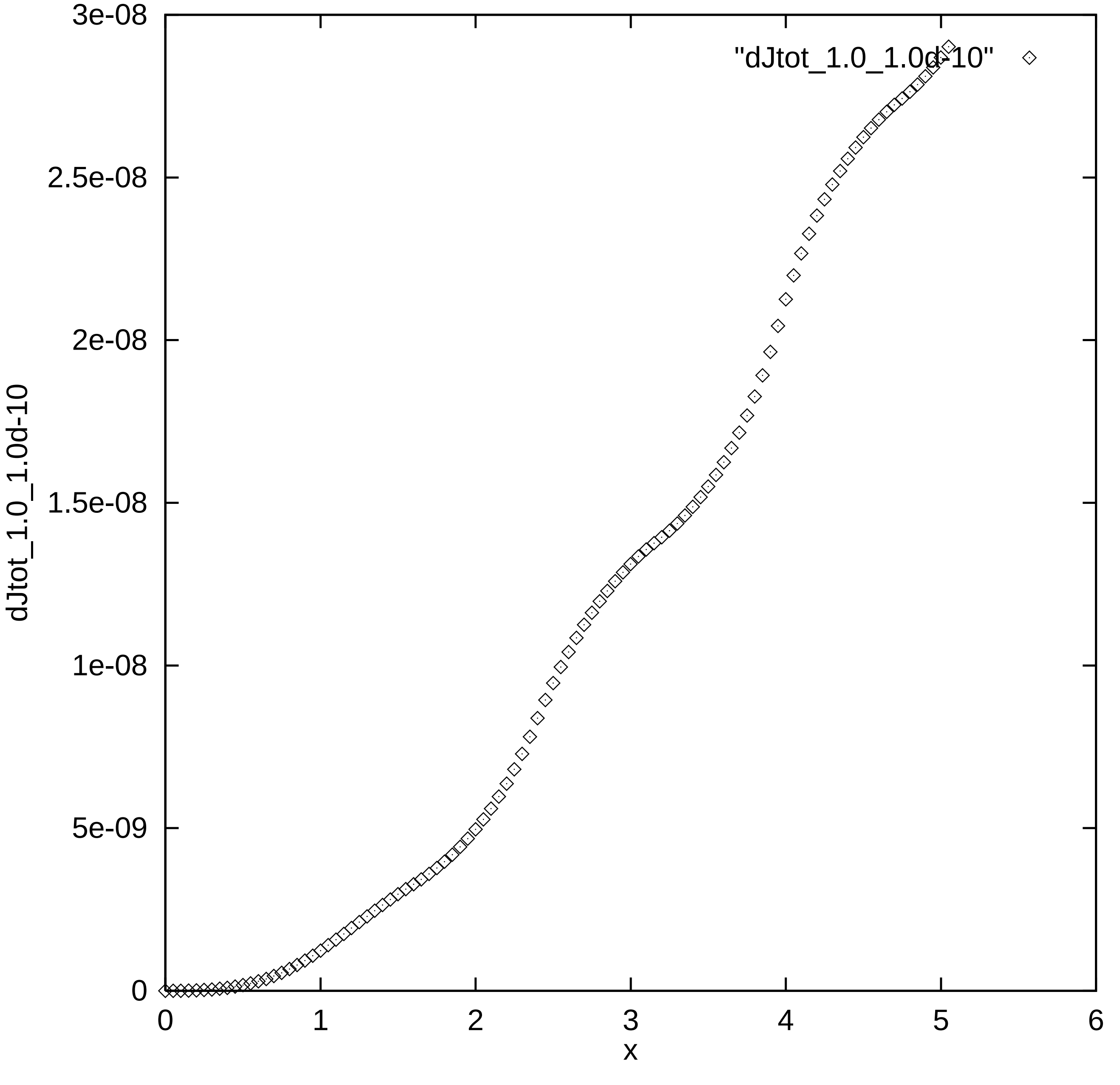

"xc_1.0_1.0d-6"
xc_1.0_1.0d-6
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6


"dEtot_1.0_1.0d-6"
dEtot_1.0_1.0d-6
x
0
-5e-05
-0.0001
-0.00015
-0.0002
-0.00025
-0.0003
0
1
2
3
4
5
6

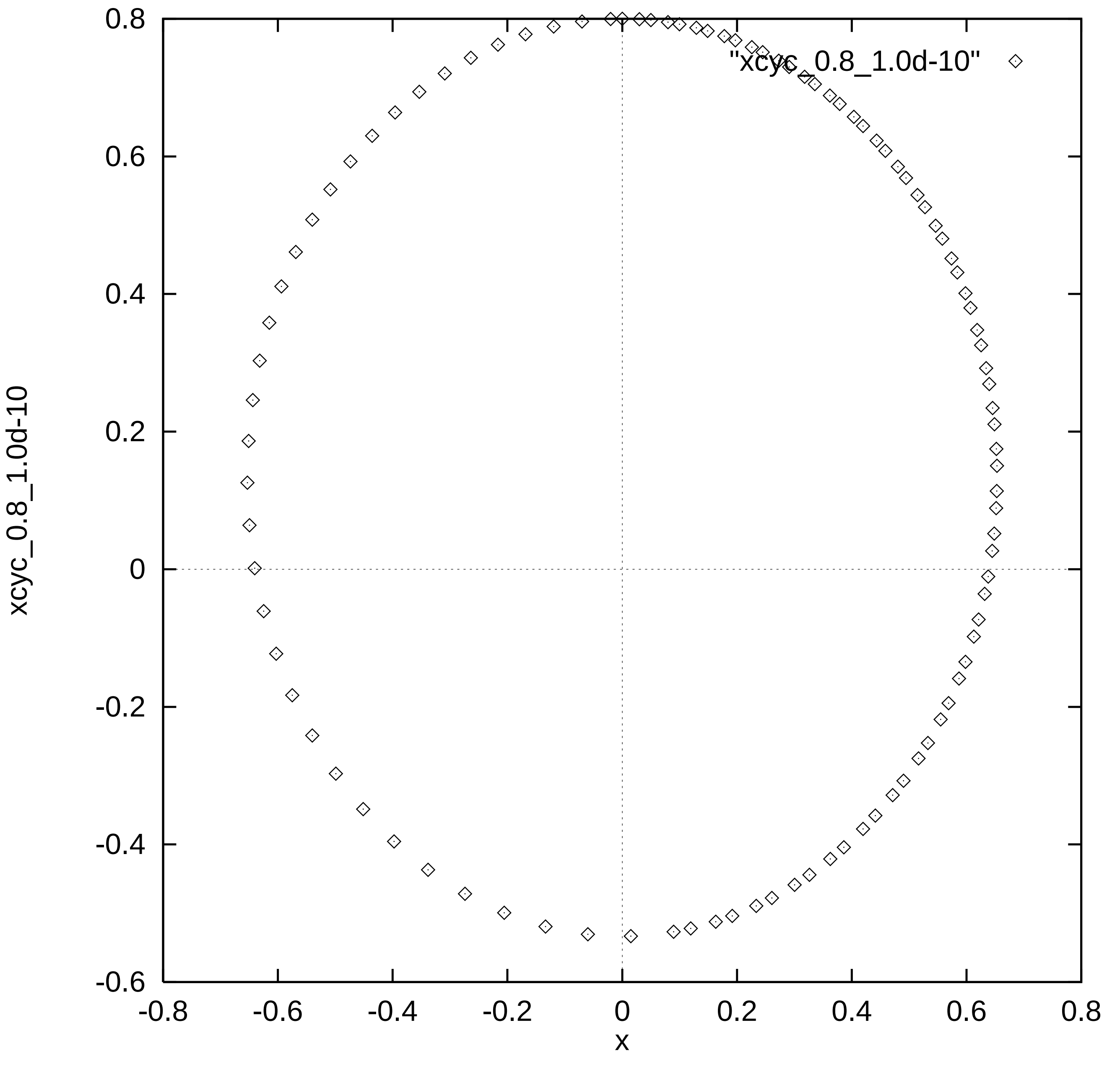


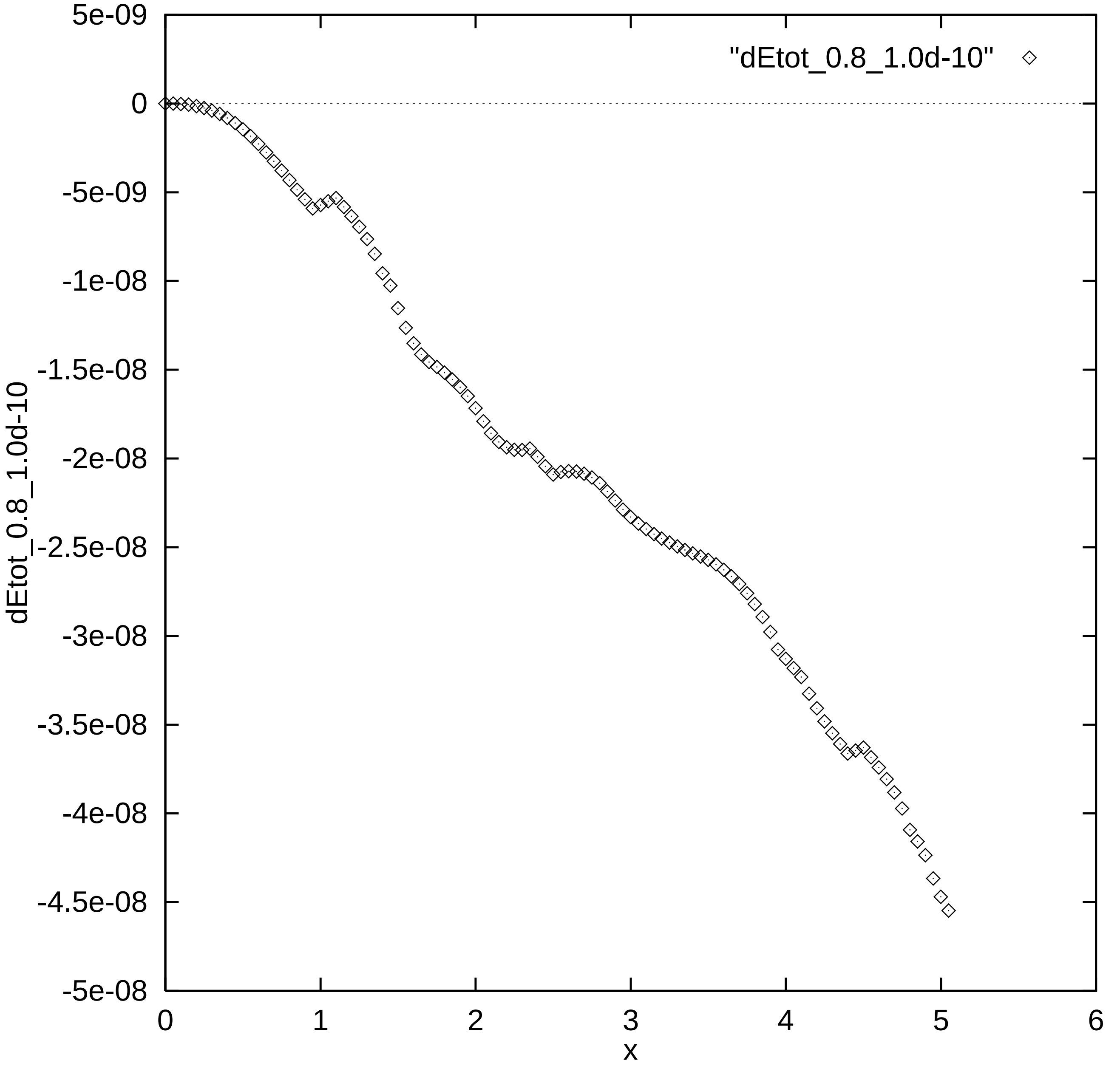
"dJtot_1.0_1.0d-6"
dJtot_1.0_1.0d-6
0.0003
0.00025
0.0002
0.00015
0.0001
5e-05
0
0
1
2
3
4
5
6
x


"dEtot_1.0_1.0d-10"
dEtot_1.0_1.0d-10
0
-5e-09
-1e-08
-1.5e-08
-2e-08
-2.5e-08
-3e-08
0
1
2
3
4
5
6
x



"dJtot_1.0_1.0d-10"
dJtot_1.0_1.0d-10
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
1
2
3
4
5
6
x


"xcyc_0.8_1.0d-10"
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
x
xcyc_0.8_1.0d-10


"dEtot_0.8_1.0d-10"
5e-09
0
-5e-09
-1e-08
-1.5e-08
-2e-08
-2.5e-08
-3e-08
-3.5e-08
-4e-08
-4.5e-08
-5e-08
dEtot_0.8_1.0d-10
0
1
2
3
4
5
6
x