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     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:
c              dtout = tmax/(2**olevel+1)
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-----------------------------------------------------------
      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        'istate' should always be set to 1 prior to
c        invocation; LSODA also uses it as a return code.
c
c        Also note that LSODA replaces 't' with the value
c        of 'tout' if the integration is successful.
c-----------------------------------------------------------
         istate = 1

         call lsoda(fcn,neq,y,t,tout,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
c-----------------------------------------------------------
c        Check return code and exit with error message if
c        there was trouble.
c-----------------------------------------------------------
         if( istate .lt. 0 ) 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`

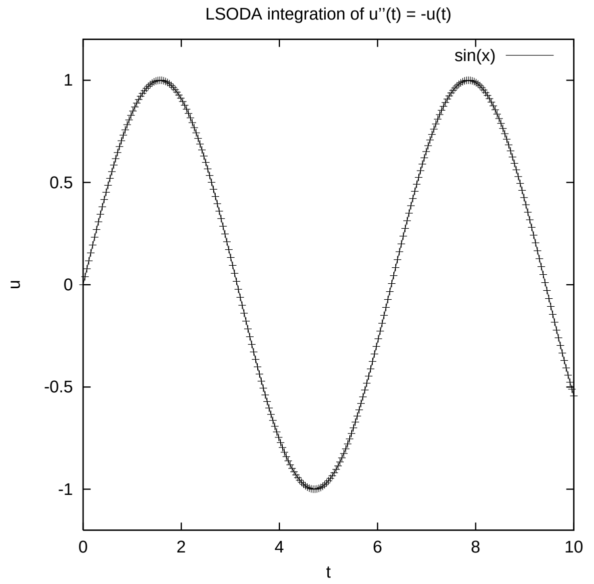
```
#!/bin/sh

#############################################################
# 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'
#############################################################

# The following disables the 'FORTRAN STOP' messages from
# PGI-compiled Fortran code.

% setenv NO_STOP_MESSAGE on

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

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

% 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       0.4948095973303711      -1.6788213251945621E-006
   0.3750000000000000       0.7325475414412526      -2.4832691574360410E-006
   0.5000000000000000       0.9588543029073683      -3.2256989623937526E-006
   0.6250000000000000        1.170198397015519      -3.8511345951106923E-006
   0.7500000000000000        1.363281824729051      -4.3046823822395854E-006
   0.8750000000000000        1.535091537001702      -4.5325296476765367E-006
    1.000000000000000        1.682946613665435      -4.6440496417296835E-006
 rmserr:   3.2594244952664999E-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:   6.6996249710096993E-005
Tolerance: 1.0e-8
 rmserr:   1.0073954017993227E-006
Tolerance: 1.0e-10
 rmserr:   1.2902520732298939E-008
Tolerance: 1.0e-12
```

```
 rmserr:   1.2675330006783407E-010

---------------------------------------------
Applying O(dt^2) approximation of ODE to
tol=1.0e-12 results
---------------------------------------------
          dt                     rms(residual)
   0.3125000000000000       5.6697977263394215E-003
   0.1562500000000000       1.4121956022936352E-003
   7.8125000000000000E-002  3.5224712324578329E-004
   3.9062500000000000E-002  8.7952460543792181E-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:   6.6996249710096993E-005
65536 output times
 rmserr:   2.8837402677146864E-004

Tolerance: 1.0e-8
No additional output times
 rmserr:   5.4600515843744528E-009
256 output times
 rmserr:   1.0073954017993227E-006
65536 output times
 rmserr:   1.1280220551945170E-004

Tolerance: 1.0e-10
No additional output times
 rmserr:   9.8365551836988008E-010
256 output times
 rmserr:   1.2902520732298939E-008
65536 output times
 rmserr:   1.1271883090994448E-006

Tolerance: 1.0e-12
No additional output times
 rmserr:   3.9105783431699081E-012
256 output times
 rmserr:   1.2675330006783407E-010
65536 output times
 rmserr:   1.5508993178092734E-008
```

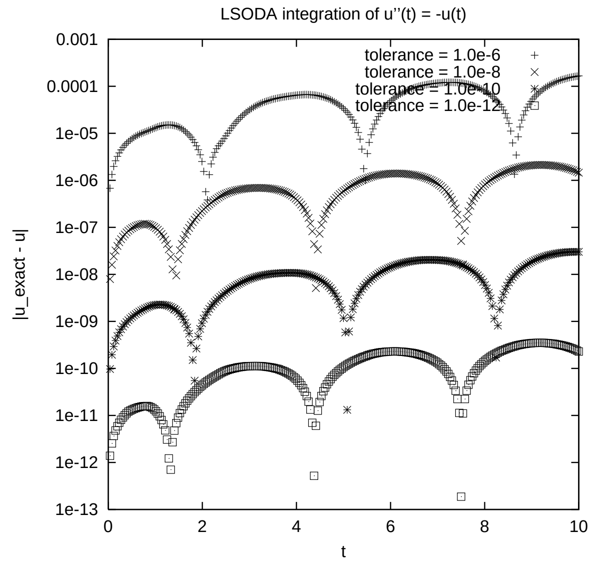
Figure file: `../tlsoda/soln.ps`



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===========================================================
      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     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 -Msecond_underscore -c integral.f
pgf77 -g -Msecond_underscore -c fcn.f
pgf77 -g -Msecond_underscore -L/usr/local/PGI/lib integral.o \
               fcn.o -lp410f -lodepack \
               -llinpack -lblas -o integral
Linking:

#############################################################
# 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 24605
#############################################################
lnx1% integral 0.0 5.0
   0.8862 2692 5446 8625
                 ^

#############################################################
# > evalf(int(exp(-x^2),x=0.0..100.0));
#
#       .8862 2692 5452 7580 1364 90835
#############################################################
lnx1% integral 0.0 100
   0.8862 2692 5446 4016
                 ^

#############################################################
# 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 2516 3508 1511
           ^
```