```fortran
c===========================================================
c     newt2:  Uses multi-dimensional Newton's method
c     to compute a root of simple non-linear system
c     discussed in class
c
c        sin(xy) - 1/2 = 0
c        y^2 - 6x - 2  = 0
c
c     Input: Command line input is initial guess (two
c     numbers) for root, and optional convergence criteria.
c
c     Output: Estimated root is written to standard output.
c
c     Tracing output: iteration number, current solution,
c     log10(|dx|), log10(|res|) written to standard error.
c===========================================================
      program        newt2

      implicit       none
c-----------------------------------------------------------
c     Functions.
c-----------------------------------------------------------
      integer        iargc
      real*8         r8arg,        drelabs,      dvl2norm
c-----------------------------------------------------------
c     For command-line parsing, an unlikely real*8 value.
c     If supplied as default to 'r8arg', return presumably
c     signals error in parsing arg. as real*8.
c-----------------------------------------------------------
      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )
c-----------------------------------------------------------
c     Command-line arguments:  Initial guess will be
c     input directly into 'x' array.
c-----------------------------------------------------------
      real*8         tol
```

```
c-------------------------------------------------------------
c     Size of system.
c-------------------------------------------------------------
      integer          neq
      parameter      ( neq = 2 )
c-------------------------------------------------------------
c     Variables used in Newton iteration and solution of
c     linear systems via LAPACK routine 'dgesv'.
c
c     J:       Jacobian matrix
c     res:     residual vector as well as solution update
c              (dgesv overwrites rhs with solution)
c     x:       Current estimate of solution vector
c     ipiv:    Pivot storage (working storage needed by dgesv)
c     info:    Return code (status code) from dgesv
c     ieq:     Local index for looping over unknowns etc.
c
c-------------------------------------------------------------
      real*8          J(neq,neq),   res(neq),
     &                x(neq)
      integer         ipiv(neq)
      integer         ieq,          info

c-------------------------------------------------------------
c     mxiter: Maximum number of Newton iterations to perform.
c             Probably wouldn't want an upper bound this high
c             in a production code.
c     nrhs:   Number of rhs's defined and sent to dgesv
c-------------------------------------------------------------
      integer         mxiter,       nrhs
      parameter     ( mxiter = 50,  nrhs = 1 )

c-------------------------------------------------------------
c     iter:    Current iteration number
c     nrm2res: l-2 norm of residual vector
c     nrm2dx:  l-2 norm of update vector
c     nrm2x:   l-2 norm of solution vector
c-------------------------------------------------------------
      integer         iter
      real*8          nrm2res,      nrm2dx,      nrm2x
```

```
c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .lt. neq ) go to 900
      do ieq = 1 , neq
         x(ieq) = r8arg(ieq,r8_never)
         if( x(ieq) .eq. r8_never ) go to 900
      end do
      tol  = r8arg(neq+1,1.0d-8)
      if( tol .le. 0.0d0 ) tol = 1.0d-8

c===========================================================
c     Newton loop.
c===========================================================
      write(0,*) 'Iter           x                     y '//
     &           '          log10(dx) log10(res)'
      write(0,*)
      do iter = 1 , mxiter
c-----------------------------------------------------------
c        Evaluate residual vector.
c-----------------------------------------------------------
         res(1) = sin(x(1)*x(2)) - 0.5d0
         res(2) = x(2)**2 - 6.0d0 * x(1) - 2.0d0
         nrm2res = dvl2norm(res,2)
c-----------------------------------------------------------
c        Set up Jacobian.
c-----------------------------------------------------------
         J(1,1) = x(2) * cos(x(1) * x(2))
         J(1,2) = x(1) * cos(x(1) * x(2))
         J(2,1) = -6.0d0
         J(2,2) = 2.0d0 * x(2)
c-----------------------------------------------------------
c        Solve linear system (J dx = res) for update
c        dx.  Update returned in 'res' vector.
c-----------------------------------------------------------
         call dgesv( neq, nrhs, J, neq, ipiv, res, neq, info )
```

```
         if( info .eq. 0 ) then
c------------------------------------------------------------
c           Update solution.
c------------------------------------------------------------
            nrm2x  = dvl2norm(x,neq)
            nrm2dx = dvl2norm(res,neq)
            do ieq = 1 , neq
               x(ieq) = x(ieq) - res(ieq)
            end do
c------------------------------------------------------------
c           Tracing output: note use of max to prevent
c           taking log10 of 0.
c------------------------------------------------------------
            write(0,1000) iter, x(1), x(2),
     &                    log10(max(nrm2dx,1.0d-60)),
     &                    log10(max(nrm2res,1.0d-60))
1000        format(i2,1p,2e24.16,0p,2f8.2)
c------------------------------------------------------------
c           Check for convergence.
c------------------------------------------------------------
            if( drelabs(nrm2dx,nrm2x,1.0d-6) .le. tol ) go to 100
         else
            write(0,*) 'newt2: dgesv failed.'
            stop
         end if
      end do
c============================================================
c     End Newton loop
c============================================================
```

```
c-----------------------------------------------------------
c     No-convergence exit.
c-----------------------------------------------------------
      write(0,*) 'No convergence after ', mxiter,
     &           ' iterations'
      stop

c-----------------------------------------------------------
c     Normal exit, write input and estimated square root
c     to standard output.
c-----------------------------------------------------------
 100  continue
      write(0,*)
      write(*,*) x
      stop

 900  continue
         write(0,*) 'usage: newt2 <x0> <y0> [<tol>]'
      stop

      end
```

```
c-----------------------------------------------------------
c     dvl2norm:  Returns l2-norm of double precision vector.
c-----------------------------------------------------------
      real*8 function dvl2norm(v,n)

         implicit      none

         integer       n
         real*8        v(n)
         integer       i

         dvl2norm = 0.0d0
         do i = 1 , n
            dvl2norm = dvl2norm + v(i) * v(i)
         end do
         if( n .gt. 0 ) then
            dvl2norm = sqrt(dvl2norm / n)
         end if

         return
      end

c-----------------------------------------------------------
c     drelabs:  Function useful for 'relativizing' quantity
c     being monitored for detection of convergence.
c-----------------------------------------------------------
      real*8 function drelabs(dx,x,xfloor)

         implicit      none

         real*8        dx,     x,    xfloor

         if( abs(x) .lt. abs(xfloor) ) then
            drelabs = abs(dx)
         else
            drelabs = abs(dx/x)
         end if

         return
      end
```

```
############################################################
#  Note that this 'Makefile' assumes that the following
#  environment variables are set:
#
#     F77
#     F77FLAGS
#     F77CFLAGS
#     F77LFLAGS
#     LIBBLAS
#     F77PP
#
#  Put the appropriate 'setenv' commands in your '~/.cshrc'.
#  See 'phy329@einstein:~/.cshrc' for an example.
############################################################
.IGNORE:

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

.f.o:
   $(F77PP) $*.f
   $(F77_COMPILE) $*.f

EXECUTABLES = newt2

all: $(EXECUTABLES)

newt2: newt2.o
   $(F77_LOAD) newt2.o -lp329f -llapack $(LIBBLAS) -o newt2

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

```
#################################################################
# Building 'newt2' and sample output on SGIs
#
# Note how different roots are found depending on the initial
# guess and how, in each case, convergence of both dx and
# the residual is quadratic as the solution is approached.
#################################################################
einstein% pwd
/Public/Members/matt/Teaching/97Spring/Phy381C/Doc/nonlin/ex3

einstein% ls *.f Makefile
Makefile   newt2.f

einstein% make -f Makefile newt2
   touch newt2.f
   f77 -g -n32 -c newt2.f
   f77 -g -n32 -L/usr/localn32/lib -n32 newt2.o \
       -lp329f -llapack -lblas -o newt2

einstein% newt2
 usage: newt2 <x0> <y0> [<tol>]

#################################################################
# Start with initial guess (1.0,1.0) and use default tolerance
#################################################################
einstein% newt2 1.0 1.0
 Iter           x                      y           log10(dx) log10(res)

 1 -3.2999966453609808E-02  1.4010001006391706E+00   -0.11    0.70
 2  3.7660093320946680E-01  2.2207017966697333E+00   -0.19   -0.40
 3  2.6508349149835875E-01  1.9187667230923000E+00   -0.64   -0.30
 4  2.7416951525985472E-01  1.9092166705387068E+00   -2.03   -1.19
 5  2.7423631305849172E-01  1.9092977465351673E+00   -4.13   -3.95
 6  2.7423631371214585E-01  1.9092977458408302E+00   -9.17   -8.33

   0.2742363137121459       1.909297745840830

#################################################################
# Start with initial guess (10.0,10.0)
#################################################################
```

```
einstein% newt2 10.0 10.0
 Iter          x                      y           log10(dx) log10(res)

 1  1.1551311217431483E+01  8.5653933652294452E+00    0.17    1.43
 2  5.2821340061715780E+00  6.2494950887336298E+00    0.67    0.26
 3  7.9156169067353002E+00  7.0845635565145253E+00    0.29    0.58
 4  8.0553488920812182E+00  7.0945184792891158E+00   -1.00   -0.08
 5  8.0478800970260274E+00  7.0913532277678391E+00   -2.24   -1.34
 6  8.0480621354266528E+00  7.0914295327798591E+00   -3.86   -2.93
 7  8.0480622340064549E+00  7.0914295740731097E+00   -7.12   -6.20

    8.048062234006455       7.091429574073110

################################################################
# Start with initial guess (100.0,100.0)
################################################################
einstein% newt2 100.0 100.0
 Iter          x                      y           log10(dx) log10(res)

 1  1.4561314470371522E+02  5.4378394341111452E+01    1.66    3.82
 2  1.9021837653952525E+02  3.7701738714769547E+01    1.53    3.17
 3  2.0349983567820649E+02  3.5070267397907138E+01    0.98    2.29
 4  2.0392234856561163E+02  3.5007684984188494E+01   -0.52    0.70
 5  2.0390326095147381E+02  3.5005993323580441E+01   -1.87   -0.53
 6  2.0391023928640129E+02  3.5006591323292412E+01   -2.31   -0.59
 7  2.0391061250942664E+02  3.5006623302706338E+01   -3.58   -1.92
 8  2.0391061457091232E+02  3.5006623479357074E+01   -5.83   -4.18
 9  2.0391061457097669E+02  3.5006623479362588E+01  -10.34   -8.68

    203.9106145709767       35.00662347936259


################################################################
# Start with initial guess (0.0,0.0), generates singular
# Jacobian
################################################################
einstein% newt2 0.0 0.0
 Iter          x                      y           log10(dx) log10(res)

 newt2: dgesv failed.
```

```
################################################################
# Start with initial guess (1.0,1.0) but use more stringent
# tolerance
################################################################
einstein% newt2 1.0 1.0 1.0e-15
 Iter           x                     y            log10(dx) log10(res)

 1 -3.2999966453609808E-02  1.4010001006391704E+00   -0.11     0.70
 2  3.7660093320946680E-01  2.2207017966697338E+00   -0.19    -0.40
 3  2.6508349149835875E-01  1.9187667230922998E+00   -0.64    -0.30
 4  2.7416951525985467E-01  1.9092166705387068E+00   -2.03    -1.19
 5  2.7423631305849172E-01  1.9092977465351673E+00   -4.13    -3.95
 6  2.7423631371214585E-01  1.9092977458408302E+00   -9.17    -8.33
 7  2.7423631371214585E-01  1.9092977458408302E+00  -60.00   -60.00

   0.2742363137121459       1.909297745840830
```

```
#################################################################
# Checking 'newt2' using numerical root finding capablities
# of Maple.
#################################################################
einstein% maple
    |\^/|     Maple V Release 3 (University of Texas - Austin)
._|\|   |/|_. Copyright (c) 1981-1994 by Waterloo Maple Software and the
 \  MAPLE  /  University of Waterloo. All rights reserved. Maple and Mapl
 <____ ____>  are registered trademarks of Waterloo Maple Software.
      |       Type ? for help.
> f1 := sin(x*y) - 1/2;

                              f1 := sin(x y) - 1/2

> f2 := y^2 - 6*x - 2;

                                       2
                               f2 := y  - 6 x - 2

> ans := fsolve({f1,f2},{x,y});

                   ans := {y = 1.909297746, x = .2742363137}

> r1 := evalf(subs(ans,f1));

                                    r1 := 0

> r2 := evalf(subs(ans,f2));

                                           -8
                                r2 := .1*10

> quit;
```

```
#################################################################
# Building 'newt2' and sample output on Cray J90
#################################################################


#################################################################
# First note that you should be able to 'rlogin' to the
# 'phas761' account on charon directly from your accounts
# on einstein.  Also note that on the SGI machines you
# can use 'charon' as an abbreviation for 'charon.cc.utexas.edu'
# with 'rlogin', 'rcp', 'telnet', 'ftp', etc.
#################################################################
einstein% rlogin charon.cc.utexas.edu -l phas761

                   The University of Texas at Austin
                  High Performance Computing Facility
        Academic Computing and Instructional Technology Services

                *** UNAUTHORIZED ACCESS PROHIBITED ***
                                 .
                                 .
                                 .

charon% ls
PHY329/    doc/       jennifer/  matt/      rmahaffy/  timbertk/
ambrose/   iliani/    lib/       metzler/   sd         xyzhang/
asbreu/    iolabar/   lischka/   phy329/    shawley/   xzhao/

charon% cd matt/nonlin
```

```
##################################################################
# Get source and Makefile from 'einstein' using 'ftp'
##################################################################
charon% ftp einstein.ph.utexas.edu
Connected to einstein.ph.utexas.edu.
220 einstein.ph.utexas.edu FTP server ready.
Remote system type is UNIX.
Using binary mode to transfer files.
Name (einstein.ph.utexas.edu:phas761): matt
331 Password required for matt.
Password:
230 User matt logged in.

ftp> cd /Public/Members/matt/Teaching/97Spring/Phy381C/Doc/nonlin/ex3
250 CWD command successful.

ftp> get newt2.f
200 PORT command successful.
150 Opening BINARY mode data connection for 'newt2.f' (7689 bytes).
226 Transfer complete.
7689 bytes received in 0.0046 seconds (1.6e+03 Kbytes/s)

ftp> get Makefile
200 PORT command successful.
150 Opening BINARY mode data connection for 'Makefile' (712 bytes).
226 Transfer complete.
712 bytes received in 0.0014 seconds (4.8e+02 Kbytes/s)

ftp> quit
221 Goodbye.

charon% ls
Makefile   newt2.f
```

```
#################################################################
# Build the application.  See ~/.cshrc on the 'phas761'
# account for setting of environment variables used in
# communication with 'make'.  In particular, note that 'F77PP'
# is set to 'f77transcray' which is a script I have written
# to translate real*8 SGI programs to single-precision
# (also 8-byte) Cray Fortran.  Be particularly careful to
# always use 'd0' notation for constants in the original (SGI)
# source.
#################################################################
charon% make
   f77transcray newt2.f
f77transcray: Translating newt2.f
   cf77 -g -c newt2.f
   cf77 -g -L/hpcf/u0/ph/as/phas761/lib newt2.o \
        -lp329f -llapack  -o newt2

#################################################################
# 'newt2' built successfully, execution on Cray is virtually
# identical to execution on SGI, except for extra messages
# written to stderr following execution of FORTRAN 'stop'.
#################################################################
charon% newt2
 usage: newt2 <x0> <y0> [<tol>]
 STOP executed at line 162 in Fortran routine 'NEWT2'
 CPU: 0.005s,  Wallclock: 0.009s,  10.5% of 5-CPU Machine
 Memory HWM: 206324, Stack HWM: 6401, Stack segment expansions: 0
```

```
####################################################################
# Start with initial guess (1.0,1.0) and use default tolerance
####################################################################
charon% newt2 1.0 1.0
 Iter           x                      y           log10(dx) log10(res)

 1 -3.2999966453608920E-02  1.4010001006391660E+00   -0.11    0.70
 2  3.7660093320946860E-01  2.2207017966697430E+00   -0.19   -0.40
 3  2.6508349149835730E-01  1.9187667230923040E+00   -0.64   -0.30
 4  2.7416951525985480E-01  1.9092166705387110E+00   -2.03   -1.19
 5  2.7423631305849300E-01  1.9092977465351660E+00   -4.13   -3.95
 6  2.7423631371214610E-01  1.9092977458408380E+00   -9.17   -8.33

 0.2742363137121462,  1.909297745840838
 STOP executed at line 158 in Fortran routine 'NEWT2'
 CPU: 0.010s,  Wallclock: 0.023s,  8.6% of 5-CPU Machine
 Memory HWM: 206327, Stack HWM: 6401, Stack segment expansions: 0
```