PHYS 410/555: Computational Physics
Homework 6 Key

Problem 1:

This problem can be solved using a straightforward modification of the source code for `new2t.f` which was covered in class. We wish to find a root of the following system of three non-linear equations:

$$\begin{aligned} f_1 &\equiv x^2 + y^3 + z^4 - 1 = 0 \\ f_2 &\equiv \sin(xyz) - (x + y + z) = 0 \\ f_3 &\equiv x - yz = 0 \end{aligned}$$

We first rewrite the system of equations in a canonical form by introducing unknowns $x_1 \equiv x$, $x_2 \equiv y$ and $x_3 \equiv z$. Then, the system becomes:

$$\begin{aligned} f_1 &\equiv {x_1}^2 + {x_2}^3 + {x_3}^4 - 1 = 0 \\ f_2 &\equiv \sin(x_1 x_2 x_3) - (x_1 + x_2 + x_3) = 0 \\ f_3 &\equiv x_1 - x_2 x_3 = 0 \end{aligned}$$

To use Newton's method, we need to evaluate the Jacobian matrix, $\mathbf{J}$, of the system. Recall that $\mathbf{J}$ has elements J_{ij} defined by

$$J_{ij} \equiv \frac{\partial f_i}{\partial x_j}$$

Thus, we have

$$\mathbf{J} = \begin{bmatrix} 2x_1 & 3{x_2}^2 & 4{x_3}^3 \\ x_2 x_3 C - 1 & x_1 x_3 C - 1 & x_1 x_2 C - 1 \\ 1 & -x_3 & -x_2 \end{bmatrix}$$

where $C \equiv \cos(x_1 x_2 x_3)$

Sample source code—`newt3.f`. See class notes for source for utility routines `drelabs` and `dvl2norm`

```
c===========================================================
c     History: <z0> newt2
c
c     newt3:  Uses multi-dimensional Newton's method
c     to compute a root of non-linear system
c
c        x^2 + y^3 + z^4 = 1
c        sin(xyz) = x + y + z
c        x = yz
c
c     Command line input is initial guess (three numbers)
c     for root, and optional convergence criteria.
c     Estimated root written to standard output.
c     Tracing output similar to that from 'newtsqrt'.
c===========================================================
      program         newt3

      implicit        none

      integer         iargc
      real*8          r8arg,        drelabs,      dvl2norm

      real*8          r8_never
      parameter     ( r8_never = -1.0d-60 )
c-----------------------------------------------------------
c     Size of system.
c-----------------------------------------------------------
      integer         neq
      parameter     ( neq = 3 )
c-----------------------------------------------------------
c     Command-line arguments:  Initial guess will be
c     input directly into 'x' array.
c-----------------------------------------------------------
      real*8          x0(neq),      tol
c-----------------------------------------------------------
c     Variables used in Newton iteration and solution of
c     linear systems via LAPACK routine 'dgesv'.
c-----------------------------------------------------------
      real*8          J(neq,neq),   res(neq),
     &                x(neq)
      integer         ipiv(neq)
      integer         ieq,          info

      integer         mxiter,       nrhs
      parameter     ( mxiter = 50,  nrhs = 1 )

      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       log10(dx) log10(res)'
      write(0,*)
      do iter = 1 , mxiter
c-----------------------------------------------------------
c        Evaluate residual vector.
c-----------------------------------------------------------
         res(1) = x(1)**2 + x(2)**3 + x(3)**4 - 1.0d0
         res(2) = sin(x(1) * x(2) * x(3)) -
     &            (x(1) + x(2) + x(3))
         res(3) = x(1) - x(2) * x(3)
         nrm2res = dvl2norm(res,neq)
c-----------------------------------------------------------
c        Set up Jacobian.
c-----------------------------------------------------------
         J(1,1) = 2.0d0 * x(1)
         J(1,2) = 3.0d0 * x(2)**2
         J(1,3) = 4.0d0 * x(3)**3
         J(2,1) = x(2) * x(3) * cos(x(1) * x(2) * x(3)) - 1.0d0
         J(2,2) = x(1) * x(3) * cos(x(1) * x(2) * x(3)) - 1.0d0
         J(2,3) = x(1) * x(2) * cos(x(1) * x(2) * x(3)) - 1.0d0
         J(3,1) = 1.0d0
         J(3,2) = -x(3)
         J(3,3) = -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,
     &                   log10(max(nrm2dx,1.0d-60)),
     &                   log10(max(nrm2res,1.0d-60))
1000        format(i2,10x,2f8.2)
c-----------------------------------------------------------
c           Check for convergence.
```

```
c-----------------------------------------------------------
            if( drelabs(nrm2dx,nrm2x,1.0d-6) .le. tol ) go to 100
         else
            write(0,*) 'newt3: dgesv failed.'
            stop
         end if
      end do
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: newt3 <x0> <y0> <z0> [<tol>]'
      stop

      end
```

Output from **newt3** when supplied with initial guess
$x = 3.0, \quad y = -2.0, \quad z = 1.0$

```
% newt3 3.0 -2.0 -1.0
Iter       log10(dx) log10(res)
 1             -0.36   -0.36
 2             -1.35   -1.35
 3             -2.29   -2.29
 4             -3.97   -3.97
 5             -7.21   -7.21
 6            -13.69  -13.69

 2.315518418880531 -1.881173775268253 -1.230890228921217
```

Problem 2:

The non-linear boundary-value problem to be solved using finite-difference techniques and Newton's method is

$$u_{xx} + (u u_x)^2 + \sin(u) = f(x) \qquad 0 \le x \le 1$$

with boundary conditions $u(0) = u(1) = 0$. As discussed in class, and adopting our usual discretization of the spatial domain (see for example, Problem 2 of Howework 4), the following is an $O(h^2)$ FDA of the continuum problem:

$$\begin{aligned} u_1 &= u(0) \\ \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + & \\ {u_i}^2 \frac{(u_{i+1} - u_{i-1})^2}{4h^2} + \sin(u_i) &= f_i \\ u_N &= u(1) \end{aligned}$$

Observe that this is a *non-linear, tri-diagonal* system of equations (which we can express as $\mathbf{F}(\mathbf{u}) = 0$) for the vector $\mathbf{u} \equiv (u_1, u_2, \cdots u_N)$. To solve this system using Newton's method, we start with some initial guess, $\mathbf{u}^{(0)}$, then generate iterates $\mathbf{u}^{(1)}$, $\mathbf{u}^{(2)}$, etc. until we have achieved convergence as discussed in the problem specification. Given an iterate $\mathbf{u}^{(n)}$, the new estimate $\mathbf{u}^{(n+1)}$ is computed using

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} - \delta \mathbf{u}^{(n)},$$

where, as usual, $\delta \mathbf{u}^{(n)}$ satisifes the *linear* equation:

$$\mathbf{J}\,\delta \mathbf{u}^{(n)} = \mathbf{r}^{(n)}$$

where $\mathbf{J}$ is the tri-diagonal Jacobian matrix with elements

$$\begin{aligned} J_{1,1} &= 1 \\ J_{1,2} &= 0 \\ J_{i,i-1} &= h^{-2}\left(1 - \frac{1}{2}u_i^2\,(u_{i+1} - u_{i-1})\right) \\ J_{i,i} &= h^{-2}\left(-2 + \frac{1}{2}u_i\,(u_{i+1} - u_{i-1})^2\right) + \cos(u_i) \\ J_{i,i+1} &= h^{-2}\left(1 + \frac{1}{2}u_i^2\,(u_{i+1} - u_{i-1})\right) \\ J_{N,N-1} &= 0 \\ J_{N,N} &= 1 \end{aligned}$$

$(i = 2, 3, \cdots N-1)$, and $\mathbf{r}^{(n)} = \mathbf{F}(\mathbf{u}^{(n)})$ is the residual vector. Observe that because the problem has Dirichlet boundary conditions, $r_1^{(n)} = r_N^{(n)} = 0$. Finally, to generate a solution $u(x) = \sin(4\pi x) \equiv \sin(\omega x)$ we must specify

$$f(x) = -\omega^2 \sin(\omega x) + \omega^2 \sin^2(\omega x)\cos^2(\omega x) + \sin(\sin(\omega x))$$

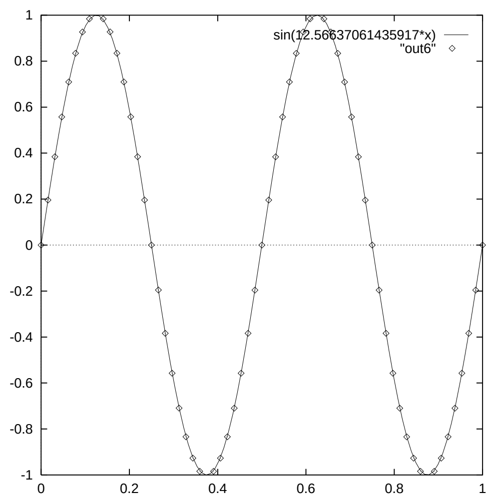
Sample source code—**nlbvp1d.f**. *Again, see class notes for source for utility routines* **drelabs** *and* **dvl2norm**

```
c===========================================================
c     Solves 1-d non-linear boundary value problem
c
c         u''(x) + (u u')^2 + sin(u) = f(x)
c
c         on  x = [0,1]; u(0) = 0, u(1) = 0
c
c     using second-order finite difference technique,
c     Newton's method and LAPACK tridiagonal solver DGTSV.
c
c     Currently set-up for solution
c
c         u(x) = sin(4 Pi x)
c===========================================================
      program        nlbvp1d

      implicit       none

      integer        i4arg
      real*8         r8arg,        drelabs,      dvl2norm

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )
c-----------------------------------------------------------
c     Extrema of problem domain.
c-----------------------------------------------------------
      real*8         xmin,         xmax
      parameter    ( xmin = 0.0d0,  xmax = 1.0d0 )
c-----------------------------------------------------------
      integer        maxn
      parameter    ( maxn = 32 769 )
c-----------------------------------------------------------
c     Storage for discrete x-values, unknowns, coefficient
c     exact solution and right hand side values.
c-----------------------------------------------------------
      real*8         x(maxn),      u(maxn),
     &               uexact(maxn), f(maxn)
c-----------------------------------------------------------
c     Storage for main, upper and lower diagonals of
c     tridiagonal system (Jacobian matrix) and
c     right-hand-side vector (residual vector) for use with
c     LAPACK routine DGTSV.  Other parameters needed for
c     call to DGTSV.
c-----------------------------------------------------------
      real*8         d(maxn),      du(maxn),
     &               dl(maxn),     rhs(maxn)
      integer        nrhs
      parameter    ( nrhs = 1 )
      integer        info

c-----------------------------------------------------------
c     Discretization level and size of system (# of discrete
c     unknowns)
c-----------------------------------------------------------
      integer        level,        n,            i,
     &               option
c-----------------------------------------------------------
c     Variables used in Newton iteration.
c-----------------------------------------------------------
      integer        mxiter
      parameter    ( mxiter = 50 )

      integer        iter
      real*8         guess_factor,               tol,
     &               nrm2res,      nrm2du,       nrm2u

c-----------------------------------------------------------
c     Enable following parameter for full tracing of
c     Newton iteration.
c-----------------------------------------------------------
      logical        newton_trace
      parameter    ( newton_trace = .false. )

c-----------------------------------------------------------
c     Mesh spacing and related constants
c-----------------------------------------------------------
      real*8         h,            hm2,          m2hm2,
     &               hm1by2,       hhm2,         qhm2
```

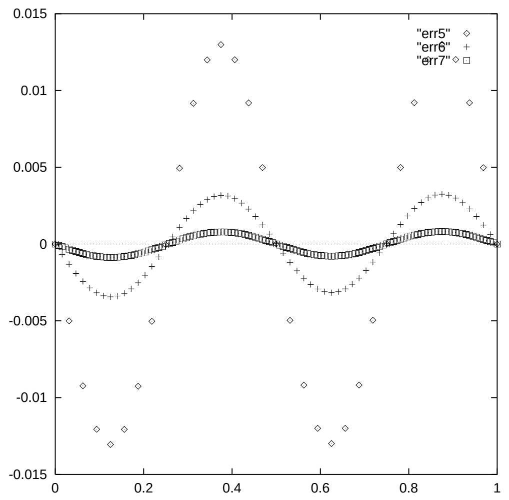
```
      real*8          rmserr
c-------------------------------------------------------------
c     Argument parsing.
c-------------------------------------------------------------
      level  = i4arg(1,-1)
      if( level .lt. 0 ) go to 900
      n = 2 ** level + 1
      if( n .gt. maxn ) then
         write(0,*) 'Insufficient internal storage'
         stop
      end if
      guess_factor = r8arg(2,r8_never)
      if( guess_factor .eq. r8_never ) go to 900
      option  = i4arg(3,0)
      tol     = r8arg(4,1.0d-8)
c-------------------------------------------------------------
c     Set up finite-difference 'mesh' (discrete x-values)
c     and define some useful constants.
c-------------------------------------------------------------
      h     =  1.0d0 / (n - 1)
      do i = 1 , n
         x(i) = xmin + (i - 1) * h
      end do
      hm2     =  1.0d0 / (h * h)
      m2hm2   = -2.0d0 / (h * h)
      hm1by2  =  0.50d0 / h
      hhm2    =  0.50d0 * hm2
      qhm2    =  0.25d0 * hm2
c-------------------------------------------------------------
c     This only ensures that x(n) = xmax EXACTLY and is not
c     essential.
c-------------------------------------------------------------
      x(n) = xmax

c-------------------------------------------------------------
c     Set up exact solution, coefficient functions and right
c     hand side vector.
c-------------------------------------------------------------
      call exact(uexact,f,x,n)

c-------------------------------------------------------------
c     Initialize unknown (u) to constant (guess_factor)
c     times exact solution.
c-------------------------------------------------------------
      do i = 1 , n
         u(i) = guess_factor * uexact(i)
      end do

c=============================================================
c     N E W T O N   L O O P
c=============================================================
      do iter = 1 , mxiter

c-------------------------------------------------------------
c        Set up tridiagonal Jacobian matrix and evaluate
c        right-hand-side (residuals)
c-------------------------------------------------------------

c-------------------------------------------------------------
c        Left boundary: Dirichlet boundary condition has
c        0 residual.
c-------------------------------------------------------------
         d(1)       = 1.0d0
         du(1)      = 0.0d0
         rhs(1)     = 0.0d0
c-------------------------------------------------------------
c        Interior: J[i,j] = d(F_i)/d_(u_j) and has non-zero
c        elements only for j = i-1, i and i+1.
c-------------------------------------------------------------
         do i = 2 , n - 1
            dl(i-1) = hm2 - hhm2 * u(i)**2 * (u(i+1) - u(i-1))
            d(i)    = m2hm2
     &                + hhm2 * u(i) * (u(i+1) - u(i-1))**2
     &                + cos(u(i))
            du(i)   = hm2 + hhm2 * u(i)**2 * (u(i+1) - u(i-1))
            rhs(i)  = hm2 * (u(i+1) - 2.0d0 * u(i) + u(i-1))
     &              + qhm2 * u(i)**2 * (u(i+1) - u(i-1))**2
     &              + sin(u(i)) - f(i)
         end do
c-------------------------------------------------------------
c        Right boundary: Dirichlet boundary condition has
c        0 residual.
c-------------------------------------------------------------
         dl(n-1)    = 0.0d0
         d(n)       = 1.0d0
         rhs(n)     = 0.0d0
c-------------------------------------------------------------
c        Compute l-2 norm of residuals.
c-------------------------------------------------------------
         nrm2res = dvl2norm(rhs,n)
         if( newton_trace ) then
            write(0,*) 'iter = ', iter
            write(0,*) 'res  = ', nrm2res
         end if
c-------------------------------------------------------------
c        Solve tridiagonal system for Newton update, delu,
c        which satisfies
c
c        J delu = residuals
c-------------------------------------------------------------

         call dgtsv( n, nrhs, dl, d, du, rhs, n, info )

         if( info .eq. 0 ) then
c-------------------------------------------------------------
c           Solver successful: compute norms of u and delu,
c           update solution and check for convergence.
c-------------------------------------------------------------
            nrm2u  = dvl2norm(u,n)
            nrm2du = dvl2norm(rhs,n)
            if( newton_trace ) then
               write(0,*) 'du   = ', nrm2du
               write(0,*) 'u    = ', nrm2u
            end if
            do i = 1 , n
               u(i) = u(i) - rhs(i)
            end do
            if( drelabs(nrm2du,nrm2u,1.0d-6) .le. tol )
     &          go to 500
         else
c-------------------------------------------------------------
c           Solver failed, write error message and exit.
c-------------------------------------------------------------
            write(0,*) 'nlbvp1d: dgtsv() failed, info = ',
     &                 info
         end if
      end do
c-------------------------------------------------------------
c     Newton iteration failed to converge: write error
c     message and exit.
c-------------------------------------------------------------
      write(0,*) 'nlbvp1d: No convergence after ', mxiter,
     &           ' iterations'
      stop

c-------------------------------------------------------------
c     Newton iteration converged: output solution or error
c     to stdout, depending on output option.  Also compute
c     rms error and output to stderr.
c-------------------------------------------------------------
500   continue

      rmserr = 0.0d0
      do i = 1 , n
         if( option .eq. 0 ) then
            write(*,*) x(i), u(i)
         else
            write(*,*) x(i), (uexact(i) - u(i))
         end if
         rmserr = rmserr + (uexact(i) - u(i)) ** 2
      end do
      rmserr = sqrt(rmserr / n)
      write(0,*) 'rmserr = ', rmserr

      stop

 900  continue
         write(0,*) 'usage: nlbvp1d <level> <guess_factor> '//
     &              '[<option> <tol>]'
         write(0,*)
         write(0,*) '       Specify option .ne. 0 for output'
```

```
         write(0,*) '        of error instead of solution'
      stop

      end

c==========================================================
c     Computes exact values for u(x) (unknown function)
c     and f(x) (right hand side function).  x array must
c     have been previously defined.
c==========================================================
      subroutine exact(u,f,x,n)

         implicit        none
         integer         n
         real*8          u(n),     f(n),      x(n)

         real*8          pi4
         integer         i

         pi4 = 16.0d0 * atan(1.0d0)
         do i = 1 , n
            u(i) = sin(pi4 * x(i))
            f(i) = -pi4**2 * sin(pi4 * x(i)) +
     &              pi4**2 * (sin(pi4 * x(i)) *
     &                        cos(pi4 * x(i)))**2 +
     &              sin(sin(pi4 * x(i)))
         end do

         return

      end
```

Level 6 solution and exact solution, `guess_factor = 1.0`



Error in level 5, 6 and 7 solutions



3 distinct solutions (level 6) found with `guess_factor` *= 0.7, 1.0 and 1.7.*