Given 3 nonlinear eqns

$$x_1^3 - e^{x_2} + \sinh x_3 = 3.6288188 \qquad (1)$$

$$x_1^2 x_3 + (x_2^2 - x_3)^2 = 4.0 \qquad (2)$$

$$x_1 x_2 x_3 - x_3 + x_1 x_2 = 5.0 \qquad (3)$$

(a) For visualization

① solve eqn 3 for x_3

$$(x_1 x_2 - 1) x_3 = 5 - x_1 x_2$$

$$\boxed{x_3 = \frac{5 - x_1 x_2}{x_1 x_2 - 1}} \qquad (4)$$

now, upon subst into eqns (1) and (2), we only have a function of two variables — x_1 and x_2

② Let's create a scalar function, $F(x_1, x_2)$, as follows

$$f_1(x_1, x_2) = x_1^3 - e^{x_2} + \sinh x_3 - 3.6288188 = 0 \qquad (5)$$

$$f_2(x_1, x_2) = x_1^2 x_3 + (x_2^2 - x_3)^2 - 4.0 = 0 \qquad (6)$$

and

$$\boxed{F(x_1, x_2) = f_1^2 + f_2^2 = 0} \qquad (7)$$

with $x_3 = f(x_1, x_2)$ as given above in eqn (4).

③ Now we can evaluate and plot $F(x_1, x_2)$ vs x_1 and x_2 in matlab to "see" approximately where this scalar function approaches zero.

(only focus on small values of $F(x_1,x_2)$)

→ Try plot3 and/or contour

(This should allow us to find some guesses for the roots)

(b) For use with fsolve we need a vector function, $\underline{f}$

$$\underline{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} x_1^3 - e^{x_2} + \sinh x_3 - 3.6288188 \\ x_1^2 x_3 + (x_2^2 - x_3)^2 - 4.0 \\ x_1 x_2 x_3 - x_3 + x_1 x_2 - 5.0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad (8)$$

→ use fsolve to find the values of x_1, x_2, x_3 where $\underline{f} = \underline{0}$

(c) For implementation of Newton's method, we will need eqn (8) as well as the Jacobian of the function.

for 3×3 system

$$\underline{\underline{J}}(\underline{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \qquad (9)$$

and for the specific function, $\underline{f}$, given by eqn (8), we have

$$\underline{\underline{J}}(\underline{x}) = \begin{bmatrix} 3x_1^2 & -e^{x_2} & \cosh x_3 \\ 2x_1 x_3 & 2(x_2^2 - x_3)(2x_2) & x_1^2 + 2(x_2^2 - x_3)(-1) \\ x_2 x_3 + x_2 & x_1 x_3 + x_1 & x_1 x_2 - 1 \end{bmatrix} \qquad (10)$$

Okay, with the above eqns, we should be able to

(a) plot the scalar function $F(x_1, x_2)$ in Matlab to identify some possible root locations

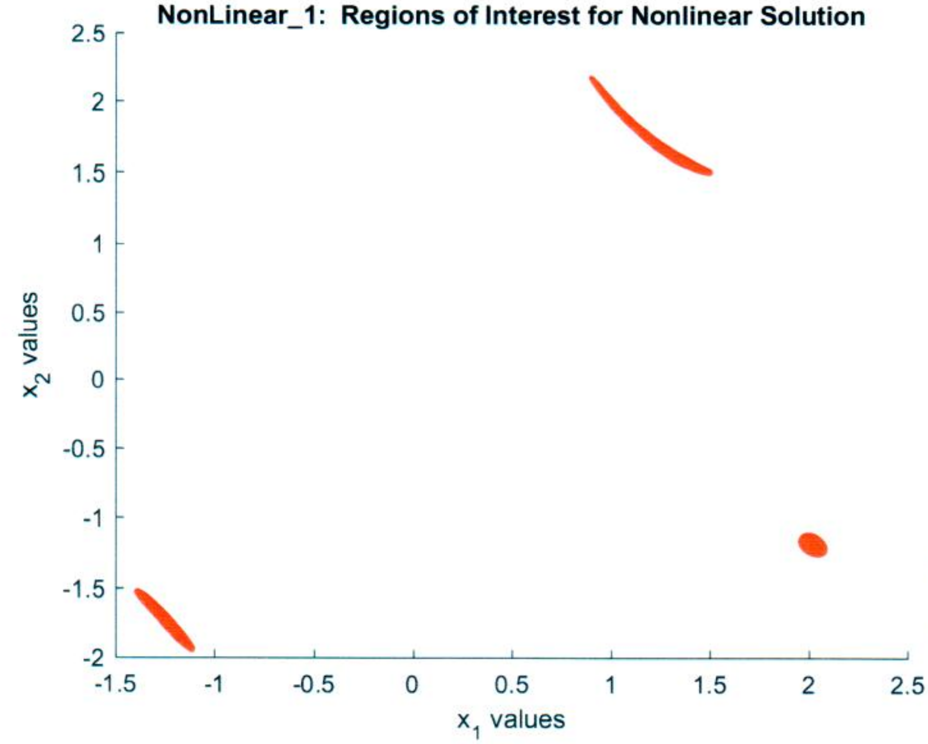
(b) + (c): Then, using these as guesses, we should be able to use Matlab's fsolve function and Newton's method to actually find the roots of the given system.

see nonlinear_1.m ← main program

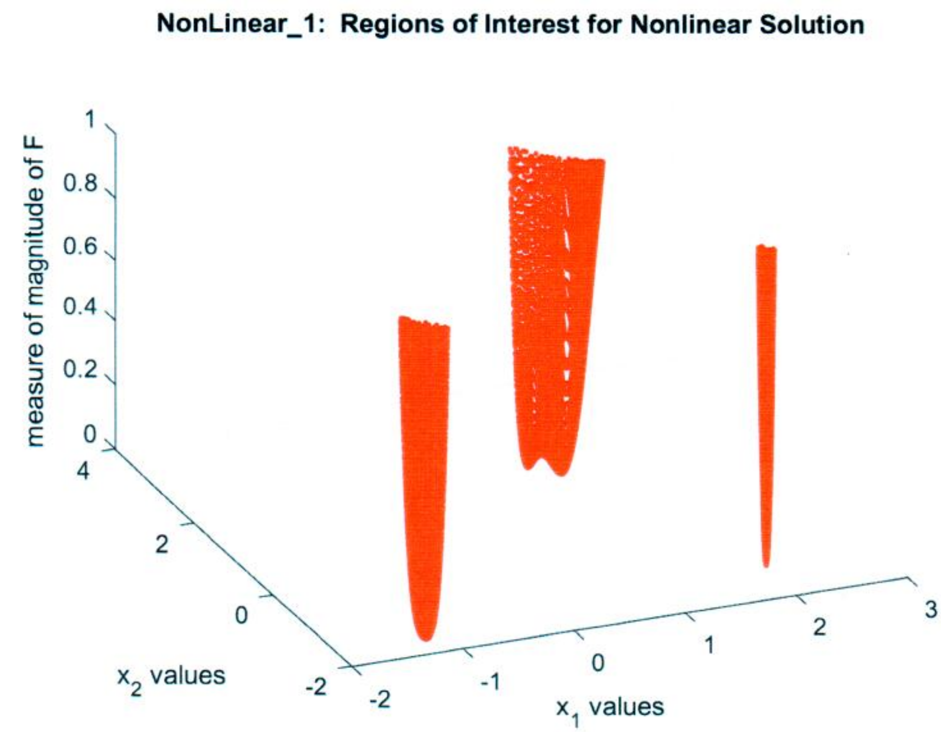
nonlinear_1a.m ← function file for use with fsolve

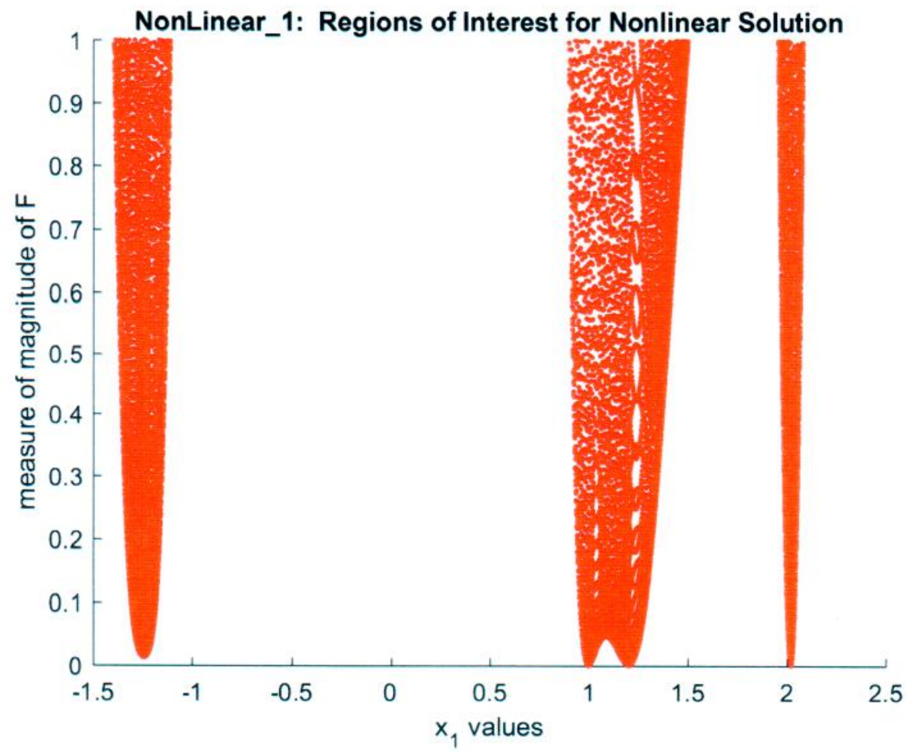
nonlinear_1newt.m ← implements Newton's method for this particular problem

Results for Nonlinear Equations (ver. 1)

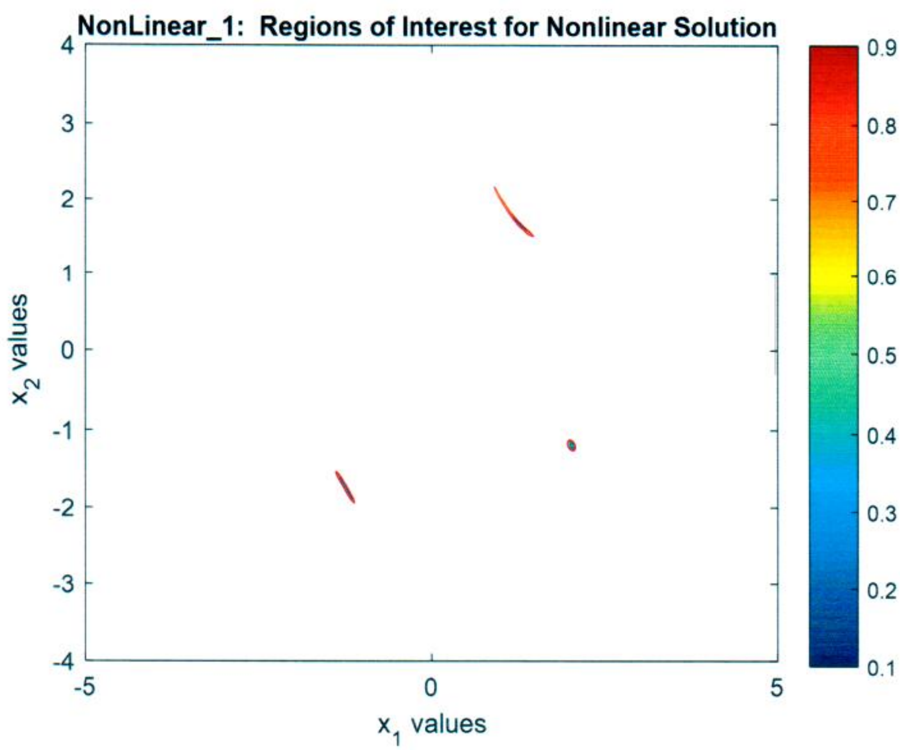


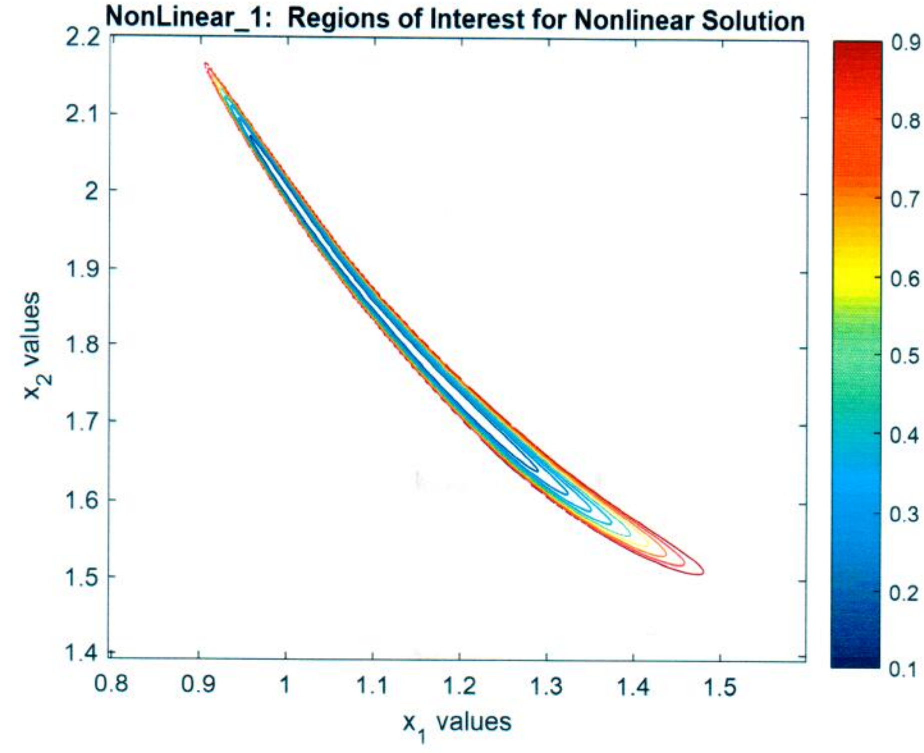
view(2)



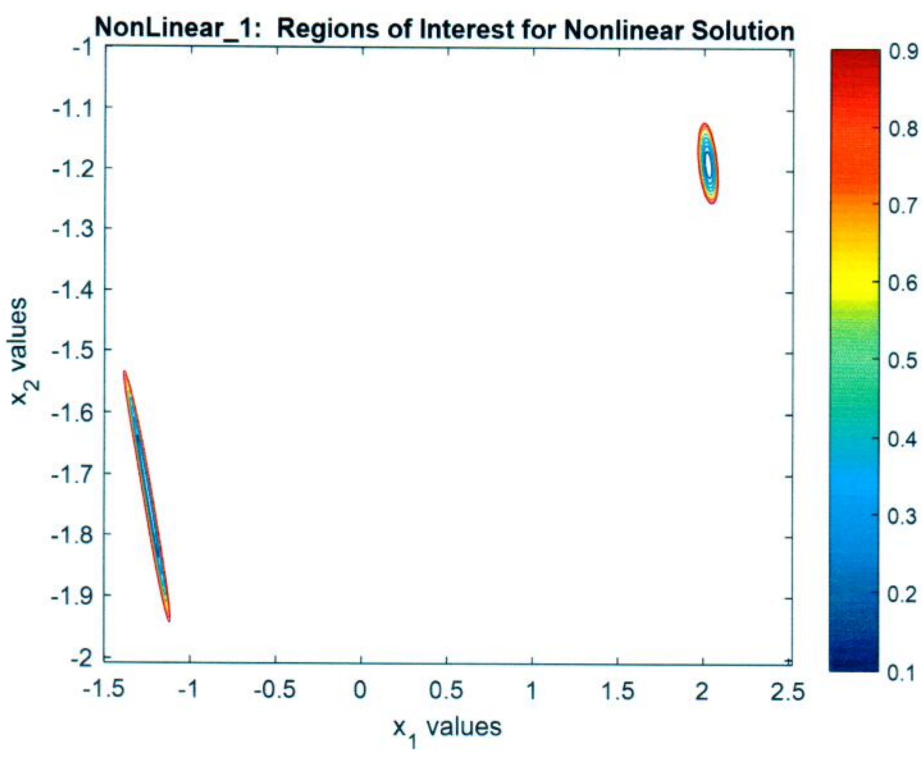

NonLinear_1: Regions of Interest for Nonlinear Solution
measure of magnitude of F
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
x_1 values

view(0,0)


NonLinear_1: Regions of Interest for Nonlinear Solution
x_2 values
4
3
2
1
0
-1
-2
-3
-4
-5
0
5
x_1 values
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1



NonLinear_1: Regions of Interest for Nonlinear Solution
2.2
2.1
2
1.9
1.8
1.7
1.6
1.5
1.4
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
x1 values
x2 values
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1



NonLinear_1: Regions of Interest for Nonlinear Solution
-1
-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
-1.8
-1.9
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
x1 values
x2 values
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

```
>> nonlinear_1

 fsolve solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    1.00     x(1) =  1.00000          f(1) =    2.8479e-08
   xo(2) =    2.00     x(2) =  2.00000          f(2) =    0.0000e+00
   xo(3) =    3.00     x(3) =  3.00000          f(3) =    0.0000e+00

 fsolve solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    1.20     x(1) =  1.20337          f(1) =    1.9642e-12
   xo(2) =    1.70     x(2) =  1.72619          f(2) =    8.1704e-12
   xo(3) =    2.85     x(3) =  2.71318          f(3) =   -2.9168e-12

 fsolve solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    2.00     x(1) =  2.02248          f(1) =   -3.9968e-15
   xo(2) =   -1.20     x(2) = -1.19022          f(2) =    3.5527e-15
   xo(3) =   -2.18     x(3) = -2.17399          f(3) =   -1.7764e-15

 WARNING --  fsolve did not converge to a root!!!  *****

 fsolve solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =   -1.20     x(1) = -1.23745          f(1) =    6.7549e-03
   xo(2) =   -1.70     x(2) = -1.73423          f(2) =    5.9384e-02
   xo(3) =    2.85     x(3) =  2.44243          f(3) =   -5.4930e-02

 Newton's method solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    1.00     x(1) =  1.00000          f(1) =    2.8479e-08
   xo(2) =    2.00     x(2) =  2.00000          f(2) =    0.0000e+00
   xo(3) =    3.00     x(3) =  3.00000          f(3) =    0.0000e+00

 Newton's method solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    1.20     x(1) =  1.20337          f(1) =    2.0829e-06
   xo(2) =    1.70     x(2) =  1.72619          f(2) =    1.8610e-06
   xo(3) =    2.85     x(3) =  2.71318          f(3) =   -1.2846e-06

 Newton's method solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =    2.00     x(1) =  2.02248          f(1) =    4.6934e-09
   xo(2) =   -1.20     x(2) = -1.19022          f(2) =    2.8270e-07
   xo(3) =   -2.18     x(3) = -2.17399          f(3) =   -9.5438e-08

*** WARNING  --  Hit max number of iterations using Newton's method!!!  ***

 Newton's method solution edit from nonlinear_1
    initial guess      solution vector     function vector at solution
   xo(1) =   -1.20     x(1) = -1.22262          f(1) =   -6.6112e-03
   xo(2) =   -1.70     x(2) = -1.76890          f(2) =    1.4091e-01
   xo(3) =    2.85     x(3) =  2.42767          f(3) =   -1.9918e-02
```

```
%
%   NONLINEAR_1.M    Solve nonlinear system using
%                Matlab's fsolve function and Newton's Method
%
%   This file computes the solution for a 3rd order nonlinear system using Matlab's
%   built-in fsolve function and a user-written file to implement Newton's method.
%     NONLINEAR_1A.m is the function file needed with fsolve
%     NONLINEAR_1NEUT.m is the function file that implements Newton's method.
%
%   The first part of this routine tries to visual the functional behavior in an
%   attempt to identify regions of interest (i.e. where the root locations are).
%
%   File prepared by J. R. White, UMass-Lowell (last update: Dec. 2017)
%

      clear all;   close all;   nfig = 0;
%
%   let's visualize regions where solutions may exist (see problem description)
      makeplot = 1;
      if makeplot == 1
      x1 = linspace(-5,5,5000);  x2 = linspace(-4,4,4000);
      [X1,X2] = meshgrid(x1,x2);
      X3 = (5.0 - X1.*X2)./(X1.*X2 - 1.0);
      F1 = X1.^3 - exp(X2) + sinh(X3) - 3.6288188;
      F2 = X1.^2.*X3 + (X2.^2 - X3).^2 - 4.0;
      F = F1.^2 + F2.^2;   F(F > 1) = nan;  % only interested in regions of small F
      nfig = nfig+1;  figure(nfig); colormap(jet)
      plot3(X1,X2,F,'r.'),grid,view(2)
      title('NonLinear\_1:  Regions of Interest for Nonlinear Solution')
      xlabel('x_1 values'),ylabel('x_2 values'),zlabel('measure of magnitude of F')
      nfig = nfig+1;  figure(nfig); colormap(jet)
      contour(X1,X2,F); grid, colorbar
      title('NonLinear\_1:  Regions of Interest for Nonlinear Solution')
      xlabel('x_1 values'),ylabel('x_2 values')

      end
%
%   from the above plot there appears to be four regions of interest, so
%   let's look at all four regions using Matlab's fsolve command (calls function
%   file nonlinear_1a.m)
      x1o = [1.0 1.2 2.0 -1.2];  x2o = [2.0 1.7 -1.2 -1.7];
      x3o = (5.0 - x1o.*x2o)./(x1o.*x2o - 1.0);
      options = optimoptions('fsolve','Display','none');
      for i = 1:length(x1o)
        xo = [x1o(i) x2o(i) x3o(i)]';
        [x,f,flag] = fsolve(@nonlinear_1a,xo,options);
        if flag ~= 1
          fprintf(1,'\n WARNING --  fsolve did not converge to a root!!!  ***** \n')
        end
        fprintf(1,'\n fsolve solution edit from nonlinear_1 \n')
        fprintf(1,'     initial guess     solution vector    function vector at ↙
solution \n')
        for j = 1:3
          fprintf(1,'    xo(%1i) = %6.2f     x(%1i) = %8.5f        f(%1i) = %12.4e\n', ↙
```

```
...
                j,xo(j),j,x(j),j,f(j))
       end
     end
%
%   now let's look at the same four starting guesses using Newton's method using
%   function file nonlinear_1neut.m
     for i = 1:length(x1o)
       xo = [x1o(i) x2o(i) x3o(i)]';
       [x,f] = nonlinear_1neut(xo);
       fprintf(1,'\n Newton''s method solution edit from nonlinear_1 \n')
       fprintf(1,'     initial guess     solution vector    function vector at ↙
solution \n')
       for j = 1:3
         fprintf(1,'    xo(%1i) = %6.2f     x(%1i) = %8.5f        f(%1i) = %12.4e\n', ↙
...
                j,xo(j),j,x(j),j,f(j))
       end
     end
%
%   Note that the last initial guess, although it looked interesting in the plots,
%   did NOT lead to a solution in either fsolve or with Newton's method.  Although
%   the function has a local minimum that approaches zero in this neighborhood, the
%   function value is not close enough to zero to meet the given convergence criteria.
%   Thus, for the x1 and x2 domains given, there are only three roots, with both the
%   fsolve and Newton methods giving the same results...
%
%   end of file
```

```
%
%   NONLINEAR_1A.M  Function file for nonlinear system (ver #1)
%                      using Matlab's fsolve command
%
%   File prepared by J. R. White, UMass-Lowell (last update: Dec. 2017)
%
      function f = nonlinear_1a(x)
      x1 = x(1);  x2 = x(2);  x3 = x(3);  % for ease in writing nonlinear eqns.
      f = [x1^3 - exp(x2) + sinh(x3) - 3.6288188;
           x1^2*x3 + (x2^2 - x3)^2 - 4.0;
           x1*x2*x3 - x3 + x1*x2 - 5.0];
%
%   end of function
```

```
%
%   NONLINEAR_1NEUT.M  Function file to use Newton's Method
%                         for nonlinear system (ver #1)
%
%   File prepared by J. R. White, UMass-Lowell (last update: Dec. 2017)
%

      function [xnew,f] = nonlinear_1neut(xold)
%
%   start iteration loop
      itmax = 50;  it = 0;  tol = 1e-6;  emax = 1;  n = length(xold);  esw = 0;
      if esw == 1, fprintf(1,'\n Intermediate edit for nonlinear_1neut  \n'), end
      while emax > tol && it <= itmax
        it = it+1;
        x = xold;
%   compute function vector using xold
       x1 = x(1);  x2 = x(2);  x3 = x(3);  % for ease in writing nonlinear eqns.
       f = [x1^3 - exp(x2) + sinh(x3) - 3.6288188;
            x1^2*x3 + (x2^2 - x3)^2 - 4.0;
            x1*x2*x3 - x3 + x1*x2 - 5.0];
%   compute Jacobian matrix evaluated at xold
        J = [ 3*x1^2      -exp(x2)        cosh(x3);
             2*x1*x3  4*(x2^2-x3)*x2  x1^2-2*(x2^2-x3);
             x2*x3+x2    x1*x3+x1        x1*x2-1];
%   compute xnew
        xnew = xold - J\f;
%   calc & edit error (intermediate results)
        emax = max(abs((xnew-xold)./xnew));
        if esw == 1
          fprintf(1,'  it = %3d      max error = %8.3e \n',it,emax)
          fprintf(1,'      xnew         xold     \n')
          for j = 1:n
            fprintf(1,' %10.5f   %10.5f    \n',xnew(j),xold(j))
          end
        end
        xold = xnew;  % use current estimate as guess for next iteration
      end
%
%   print final max relative error and iteration count
      if esw == 1
        fprintf(1,'\n Number of iterations to convergence = %3d\n',it)
        fprintf(1,' Max relative error at convergence =  %8.3e\n',emax)
      end
      if it >= itmax
        fprintf(1,'\n ***** WARNING  --  Hit max number of iterations using Newton''s ↙
method!!!  *****\n')
      end
%
%   end of function
```