Given:

$$f_1 = x_1^2 - 2x_1 - x_2 + 0.5$$

$$f_2 = x_1^2 + 4x_2^2 - 4.0$$

Then

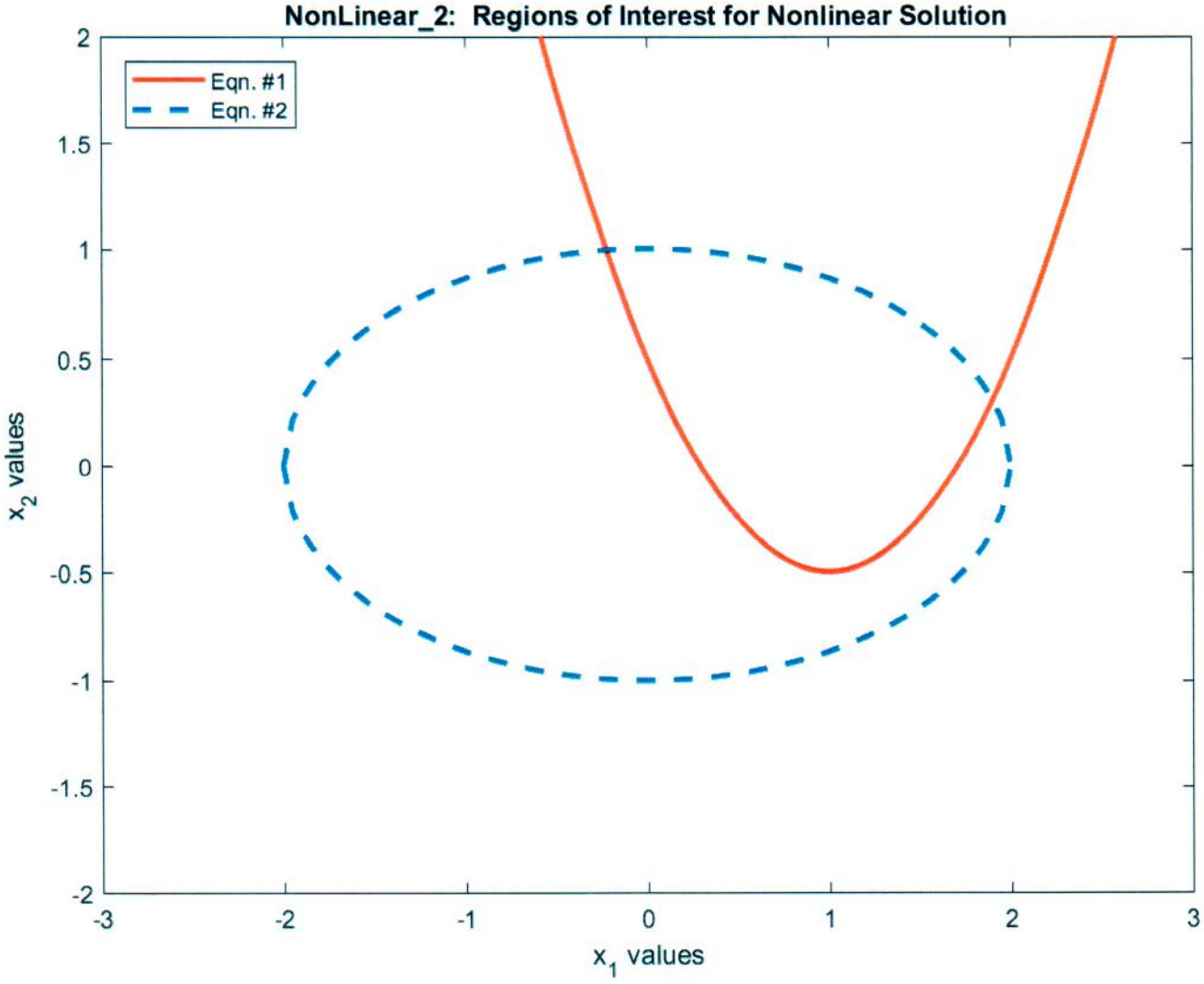
$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}$$

or

$$J = \begin{bmatrix} 2x_1 - 2 & -1 \\ 2x_1 & 8x_2 \end{bmatrix}$$

ans

this will be used within nonlinear_2 newton.m


NonLinear_2: Regions of Interest for Nonlinear Solution
Eqn. #1
Eqn. #2
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-3
-2
-1
0
1
2
3
x2 values
x1 values

```
>> nonlinear_2

fsolve solution edit from nonlinear_2
    initial guess     solution vector     function vector at solution
   xo(1) =   1.90     x(1) =  1.90068        f(1) =   3.8676e-09
   xo(2) =   0.30     x(2) =  0.31122        f(2) =   5.3574e-08


fsolve solution edit from nonlinear_2
    initial guess     solution vector     function vector at solution
   xo(1) =  -0.20     x(1) = -0.22221        f(1) =   5.4947e-08
   xo(2) =   1.00     x(2) =  0.99381        f(2) =   7.4058e-08


Newton's method solution edit from nonlinear_2
    initial guess     solution vector     function vector at solution
   xo(1) =   1.90     x(1) =  1.90068        f(1) =   3.8666e-09
   xo(2) =   0.30     x(2) =  0.31122        f(2) =   5.3572e-08


Newton's method solution edit from nonlinear_2
    initial guess     solution vector     function vector at solution
   xo(1) =  -0.20     x(1) = -0.22221        f(1) =   5.4943e-08
   xo(2) =   1.00     x(2) =  0.99381        f(2) =   7.4056e-08
```

```
%
%   NONLINEAR_2.M    Solve nonlinear system using
%              Matlab's fsolve function and Newton's Method
%
%   This file computes the solution for a 2nd order nonlinear system using Matlab's
%   built-in fsolve function and a user-written file to implement Newton's method.
%
%   NONLINEAR_2NEUT.m is the function file that implements Newton's method.
%
%   File prepared by J. R. White, UMass-Lowell (last update: Dec. 2017)
%

  clear all;   close all;   nfig = 0;
%
%   let's visualize the intersection points where solutions may exist
  x1 = linspace(-3,3,121);
  x2_1 = x1.*x1-2*x1+0.5;
  x2_2 = sqrt(1-x1.*x1/4);
  x2_2(abs(imag(x2_2))>0) = nan;  % eliminates complex values
  nfig = nfig+1;  figure(nfig);
  plot(x1,x2_1,'r-',x1,x2_2,'b--',x1,-x2_2,'b--','LineWidth',2),grid on
  title('NonLinear\_2:  Regions of Interest for Nonlinear Solution')
  xlabel('x_1 values'),ylabel('x_2 values')
  rr = axis;  rr(4) = 2;  axis(rr);
  legend('Eqn. #1','Eqn. #2','Location','NorthWest')
%
%   from the above plot there are two regions of interest, so let's look at each
%   of these using Matlab's fsolve command (uses anonymous function for f(x))
  fx = @(x) [(x(1)^2 - 2*x(1) - x(2) + 0.5); (x(1)^2 + 4*x(2)^2 - 4.0)];
  x1o = [1.9 -0.2];  x2o = [0.3 1.0];
  options = optimoptions('fsolve','Display','none');
  for i = 1:length(x1o)
    xo = [x1o(i) x2o(i)]';
    [x,f,flag] = fsolve(fx,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_2 \n');
    fprintf(1,'     initial guess     solution vector    function vector at solution↙
\n');
    for j = 1:2
      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
    fprintf(1,'\n');
  end
%
%   now let's look at the same starting guesses using Newton's method using function
%   file nonlinear_2neut.m
  for i = 1:length(x1o)
    xo = [x1o(i) x2o(i)]';
    [x,f] = nonlinear_2neut(xo);
    fprintf(1,'\n Newton''s method solution edit from nonlinear_2 \n');
    fprintf(1,'     initial guess     solution vector    function vector at solution↙
\n');
```

```
      for j = 1:2
        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
      fprintf(1,'\n');
    end
%
%   end of file
```

```
%
%   NONLINEAR_2NEUT.M  Function file to use Newton's Method
%                        for nonlinear system (ver #2)
%
%   File prepared by J. R. White, UMass-Lowell (last update: Dec. 2017)
%

    function [xnew,f] = nonlinear_2neut(xold)
%
%   start iteration loop
      itmax = 30;  it = 0;  tol = 1e-6;  emax = 1;  n = length(xold);
      esw = 0;  % edit switch (0 gives no extra edit  or  1 gives intermediate edit)
      if esw == 1, fprintf(1,'\n Intermediate edit for nonlinear_2neut  \n'); end
      while emax > tol && it <= itmax
        it = it+1;
        x = xold;
%     compute function vector using xold
       x1 = x(1);  x2 = x(2);   % for ease in writing nonlinear eqns.
       f = [x1^2 - 2*x1 - x2 + 0.5;
            x1^2 + 4*x2^2 - 4.0];
%     compute Jacobian matrix evaluated at xold
        J = [ 2*x1-2     -1;
              2*x1      8*x2];
%     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,' ***** WARNING  --  Hit max number of iterations!!!  *****\n');
      end
%
%   end of function
```