MATLAB® files for test of Newton's method for 2 nonlinear equations with a solution at (3,4)

This program uses the routine reduced_Newton.m listed separately on the "Lecture Material" web page.

All code generated with Matlab® Software

File: Newton_2D_test2.m

```
% Newton_2D_test2.m
% This m-file investigates the use of a reduced Newton's step
% method to solve a set of two nonlinear equations.
% K. Beers. MIT ChE. 10/17/2001


% First, make a filled 2-D contour plot of the norm of the function vector.

x1_min = -5;
x1_max = 5;
Npts_x1 = 50;
x1_vect = linspace(x1_min,x1_max,Npts_x1);
delta_x1 = x1_vect(2)-x1_vect(1);

x2_min = -5;
x2_max = 5;
Npts_x2 = 50;
x2_vect = linspace(x2_min,x2_max,Npts_x2);
delta_x2 = x2_vect(2)-x2_vect(1);



% Make 2D grid of mesh points.

[X1,X2] = meshgrid(x1_vect,x2_vect);
F1 = 3*X1.^3 + 4*X2.^2 - 145;
F2 = 4*X1.^2 - X2.^3 + 28;
Fmax = max(abs(F1),abs(F2));


% Now, make a coarse grid for the gradient plots.

num_coarse = 10;
scale_quiver = 0.5;

x1_c = linspace(x1_min,x1_max,num_coarse);
delta_x1_c = x1_c(2)-x1_c(1);

x2_c = linspace(x2_min,x2_max,num_coarse);
```

```
delta_x2_c = x2_c(2)-x2_c(1);

[X1_c,X2_c] = meshgrid(x1_c,x2_c);

F1_c = 3*X1_c.^3 + 4*X2_c.^2 - 145;
F2_c = 4*X1_c.^2 - X2_c.^3 + 28;
Fmax_c = max(abs(F1_c),abs(F2_c));


[DF1_DX1,DF1_DX2] = gradient(F1_c,delta_x1_c,delta_x2_c);
[DF2_DX1,DF2_DX2] = gradient(F2_c,delta_x1_c,delta_x2_c);
[DFmax_DX1,DFmax_DX2] = gradient(Fmax_c,delta_x1_c,delta_x2_c);
```

% Now make the plots.

```
figure;
hold on;
contour(X1,X2,F1,20);
quiver(X1_c,X2_c,DF1_DX1,DF1_DX2,scale_quiver);
title('f_1(x_1,x_2) = 3*x_1^2 + 4*x_2^2 - 145');
xlabel('x_1'); ylabel('x_2');

figure;
hold on;
contour(X1,X2,F2,20);
quiver(X1_c,X2_c,DF2_DX1,DF2_DX2,scale_quiver);
title('f_2(x_1,x_2) = 4*x_1^2 - x_2^3 + 28');
xlabel('x_1'); ylabel('x_2');

figure;
hold on;
contour(X1,X2,Fmax,20);
quiver(X1_c,X2_c,DFmax_DX1,DFmax_DX1,scale_quiver);
title('||f|| = max(|f_1|, |f_2|)');
xlabel('x_1'); ylabel('x_2');
```

% ==========

% We now plot a trajectory for the Newton's method iterations.

```
max_iter = 100;
x_guess = input('Input initial guess as column vector : ');

calc_f = 'Newton_2D_test1b_calc_f';
calc_Jac = 'Newton_2D_test1b_calc_Jac';
```

% ----------

```
% First, use full Newton's method.

   x_traj = zeros(max_iter+1,2);
   x = x_guess;

   k = 1;
   x_traj(k,:) = x';

   for j=1:max_iter

      k = k + 1;

      % calculate Jacobian matrix
      Jac=zeros(2,2);
      Jac(1,1) = 9*x(1)^2;
      Jac(1,2) = 8*x(2);
      Jac(2,1) = 8*x(1);
      Jac(2,2) = -3*x(2)^2;

      % calculate function vector
      f = [0;0];
      f(1) = 3*x(1)^3 + 4*x(2)^2 - 145;
      f(2) = 4*x(1)^2 - x(2)^3 + 28;

      % check for convergence
      f_norm = max(abs(f));
      if(f_norm < 1.e-10)
         iter_conv = j;
         break;
      end

      % solve for Newton step update
      dx = Jac\(-f);

      % update solution estimate
      x = x + dx;

      % store new result in trajectory vector
      x_traj(k,:) = x';

   end


   % Make plot of trajectory over that of norm of f.

   figure;
   hold on;
   contour(X1,X2,Fmax,20);
   quiver(X1_c,X2_c,DFmax_DX1,DFmax_DX2,scale_quiver);
   title('Newton''s method, ||f||, trajectory');
   xlabel('x_1'); ylabel('x_2');
```

```
   if(iter_conv)
      plot(x_traj(1:iter_conv,1),x_traj(1:iter_conv,2),'-o');
      gtext(['Initial guess : x_1 = ', num2str(x_guess(1)), ...
         ', x_2 = ', num2str(x_guess(2)), ...
         '  # iter. = ', int2str(iter_conv)]);
   else
      plot(x_traj(:,1),x_traj(:,2),'-o');
       gtext(['Initial guess : x_1 = ', num2str(x_guess(1)), ...
         ', x_2 = ', num2str(x_guess(2)), ...
         '  unconverged']);
   end

   gtext('f_1 = 3*x_1^2 + 4*x_2^2 - 145, f_2 = 4*x_1^2 - x_2^3 + 28');

   xfull_traj = x_traj;
   iter_full_conv = iter_conv;



% ------------------------------------------------------------
% Next, use the reduced Newton's step method with the same
% convergence criterion.

Options.max_iter = max_iter;
Options.max_iter_LS = 10000;
Options.rtol = 1;
Options.atol = 1.e-10;
Options.step_tol = 1.e-3;
Options.verbose = 1;
Options.use_range = 1;
Options.range = [10; 10];
Param = 0;
x0 = x_guess;

[x,iflag,iter_conv,x_traj,f_traj] = ...
   reduced_Newton(x0,calc_f,calc_Jac,Options,Param);


% Now we plot the reduced Newton method trajectory.

figure;
hold on;
contour(X1,X2,Fmax,20);
quiver(X1_c,X2_c,DFmax_DX1,DFmax_DX2,scale_quiver);
title('Reduced-step Newton''s method, ||f||, trajectory');
xlabel('x_1'); ylabel('x_2');

if(iter_conv)
   plot(x_traj(:,1),x_traj(:,2),'-o');
   gtext(['Initial guess : x_1 = ', num2str(x_guess(1)), ...
```

```
       ', x_2 = ', num2str(x_guess(2)), ...
       '  # iter. = ', int2str(iter_conv)]);
else
   plot(x_traj(:,1),x_traj(:,2),'-o');
     gtext(['Initial guess : x_1 = ', num2str(x_guess(1)), ...
       ', x_2 = ', num2str(x_guess(2)), ...
       '  unconverged']);
end
gtext('f_1 = 3*x_1^2 + 4*x_2^2 - 145, f_2 = 4*x_1^2 - x_2^3 + 28');
```

File: Newton_2D_test1b_calc_f.m

```
% Newton_2D_test1b_calc_f.m
%
% This MATLAB m-file calculates the function value
% for a simple set of 2 non-linear equations.
%
% K. Beers. MIT ChE. 10/17/2001


function f = Newton_2D_test1b_calc_f(x,Param);

f = [0; 0];

f(1) = 3*x(1)^3 + 4*x(2)^2 - 145;
f(2) = 4*x(1)^2 - x(2)^3 + 28;

return;
```

File: Newton_2D_test1b_calc_Jac.m

```
% Newton_2D_test1b_calc_Jac.m
%
% This MATLAB m-file calculates the Jacobian
% for a simple set of 2 non-linear equations.
%
% K. Beers. MIT ChE. 10/17/2001


function Jac = Newton_2D_test1b_calc_Jac(x,Param);

Jac = zeros(2,2);

Jac(1,1) = 9*x(1)^2;
Jac(1,2) = 8*x(2);
Jac(2,1) = 8*x(1);
Jac(2,2) = -3*x(2)^2;


return;
```