```
% Specify mesh x and initial guess U0
function U = ...
    Chap5_CalculateFemForNonlinearBVP_Jacobian(x, U0)

% Use Matlab routine to solve nonlinear algebraic system

% Specify that the Jacobian will be specified by the user
options = optimoptions('fsolve','Jacobian','on');

U = fsolve(@CalculateResidualAndJacobian, U0, options, x);

% Plot solution
plot(x, U, '-')
xlabel('x')
ylabel('U')



% Function for calculating global residual and Jacobian
function [Residual, Jacobian] = ...
    CalculateResidualAndJacobian(U, x)

% Deduce number of elements
N = length(x) - 1;

% Initialise residual and Jacobian to zero
Residual = zeros(N+1, 1);
Jacobian = zeros(N+1, N+1);

% Loop over elements, computing local contribution to
% residual and incrementing global residual and Jacobian
for k=1:N
    [LocalResidual, LocalJacobian] = ...
        CalculateResidualAndJacobianOnElement(U, x, k);
    Residual(k:k+1) = Residual(k:k+1) + LocalResidual;
    Jacobian(k:k+1, k:k+1) = Jacobian(k:k+1, k:k+1) + ...
        LocalJacobian;
end

% Handle Neumann boundary condition at x=2
Residual(N+1) = Residual(N+1) - 68;

% Handle Dirichlet boundary condition at x=-1
Residual(1) = U(1) - 1;
Jacobian(1,:) = 0;
Jacobian(1,1) = 1;



% Function for calculating local contribution to
% residual and Jacobian
function [LocalResidual, LocalJacobian] = ...
    CalculateResidualAndJacobianOnElement(U, x, k)

% Set element length h
h = x(k+1) - x(k);

% Define quadrature rule
```

```
M = 3;
QuadWeights = [5/18, 4/9, 5/18];
QuadPoints = [0.5*(1-sqrt(0.6)), 0.5, 0.5*(1+sqrt(0.6))];

% Initialise local residual and Jacobian to zero
LocalResidual = zeros(2,1);
LocalJacobian = zeros(2,2);

% Loop over quadrature points to evaluate integral
% using Gaussian quadrature
for m=1:M
   % Set local coordinate X
   X = QuadPoints(m);

   % Calculate local values of basis functions,
   % their derivatives, x, U and the derivative of U
   phi_local = [1-X, X];
   phi_prime_local = [-1, 1];
   x_local = x(k) + h*X;
   LocalU = U(k)*phi_local(1) + U(k+1)*phi_local(2);
   LocalUPrime = U(k)*phi_prime_local(1) + ...
      U(k+1)*phi_prime_local(2);

   % Evaluate integrand and add into sum
   for i=1:2
       LocalResidual(i) = LocalResidual(i) + ...
          h*QuadWeights(m)* ...
          (1/h/h*(1+LocalU*LocalU)* ...
          LocalUPrime*phi_prime_local(i) + ...
          2*(1+x_local^4)*phi_local(i) + ...
          8*LocalU*LocalU*phi_local(i));

      for j=1:2
           LocalJacobian(i,j) = LocalJacobian(i,j) + ...
               h*QuadWeights(m)* ...
               (1/h/h*(2*LocalU*phi_local(j)* ...
               LocalUPrime + ...
               (1+LocalU*LocalU)*phi_prime_local(j))* ...
               phi_prime_local(i) + ...
               16*LocalU*phi_local(j)*phi_local(i));
      end

   end
end
```