```
function [] = thermal_problem_quadratic()
    num_elem = 16;
    num_node = 2*num_elem + 1;
    L = 0.5;        % m
    kappa = 200;    % W/m K
    h = 1000;       % W/m^2 K
    A = 0.020^2;    % m^2
    mesh.x = linspace(0, L, num_node);
    mesh.conn = [1:2:num_node-1; 2:2:num_node; 3:2:num_node];
    % This is an anonymous function - see link below for more details.
    % http://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html
    s = @(x) 1000*(L-x)*x^2;
    K = zeros(num_node, num_node);
    f = zeros(num_node, 1);
    % In this loop we compute the conductivity and nodal flux matrices.
    for c = mesh.conn
        xe = mesh.x(c);
        Ke = zeros(3);
        for q = quadrature(2)
            N = shape(q);
            dNdp = gradshape(q);
            Je = xe * dNdp;
            dNdx = dNdp / Je;
            Ke = Ke + dNdx*kappa*A*dNdx'*Je*q(2);
            x = xe*N;
            f(c) = f(c) + N*s(x) * Je*q(2);
        end
        K(c,c) = K(c,c) + Ke;
    end
    % Sets temperature-dependent flux boundary condition.
    f(end) = f(end) + A*30*h;
    K(end,end) = K(end,end) + A*h;

    % Sets fixed temperature on left node (to 30 deg C).
    K(1,:) = 0;
    K(1,1) = 1;
    f(1) = 30;
    % Solves the system of equations.
    d = K\f;

    % Post processing: interpolates temperature and heat flux within
    % elements.
    pos = []; temp = []; flux = [];
    for c = mesh.conn
        xe = mesh.x(c);
        de = d(c)';
        for q = linspace(-1, 1, 10)
            N = shape(q);
            dNdp = gradshape(q);
            Je = xe * dNdp;
            dNdx = dNdp / Je;
            pos(end+1) = xe*N;
            temp(end+1) = de*N;
            flux(end+1) = -kappa*de*dNdx;
        end
    end
    subplot(2,1,1);
    plot(pos, temp, 'b-', mesh.x, d, 'bo');
    xlabel('position');     ylabel('temperature');
    subplot(2,1,2);
    plot(pos, flux, 'r');
    xlabel('position');     ylabel('heat flux');
end

function [N] = shape(q)
    N = [0.5*q(1)*(q(1)-1); 1-q(1)^2; 0.5*q(1)*(1+q(1))];
end
function [dNdp] = gradshape(q)
    dNdp = [q(1)-0.5; -2*q(1); q(1)+0.5];
end
function [quad_pts] = quadrature(n)
    if n == 1
        quad_pts = [0; 2];
    elseif n == 2
        quad_pts = [[-1, 1]/sqrt(3); 1, 1];
    elseif n == 3
        quad_pts = [[1,0,-1]*sqrt(3/5); [5,8,5]/9];
```

```
    elseif n == 4
        x1 = sqrt(3/7 - 2/7*sqrt(6/5));
        x2 = sqrt(3/7 + 2/7*sqrt(6/5));
        w1 = (18+sqrt(30)) / 36;
        w2 = (18-sqrt(30)) / 36;
        quad_pts = [x1,-x1, x2, -x2; w1, w1, w2, w2];
    else
        error('Quadrature rule n must be 1, 2, 3, or 4');
    end
end
```