```
function [x, U] = ...
    Chap4_CalculateQuadraticFemForBvp(x_ele_bound)

% define p(x), q(x), r(x), f(x), u_a, eta_b
p = @(x) -x;
q = @(x) -(2*x+3);
r = @(x) x+2;
f = @(x) x^3*exp(2*x);
u_a = exp(1);
eta_b = -5*exp(1)*exp(1);

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

% Allocate memory for nodes
x = zeros(2*N+1, 1);

% Define nodes on edge of elements
x(1:2:2*N+1) = x_ele_bound;

% Define nodes at centre of elements
x(2:2:2*N) = 0.5*(x(1:2:2*N-1) + x(3:2:2*N+1));

% Initialise A and b to zero
A = zeros(2*N+1, 2*N+1);
b = zeros(2*N+1, 1);

% Loop over elements calculating local contributions and
% inserting into linear system
for k = 1:N
    [A_local, b_local] = ...
        CalculateContributionOnElement(x, k, p, q, r, f);

    A(2*k-1:2*k+1, 2*k-1:2*k+1) =  ...
        A(2*k-1:2*k+1, 2*k-1:2*k+1) + A_local;
    b(2*k-1:2*k+1) =  b(2*k-1:2*k+1) + b_local;
end

% Add contributions from Neumann boundary conditions into
% linear system
b(2*N+1) = b(2*N+1) + eta_b;

% Set Dirichlet boundary conditions in row 1
A(1,:) = 0;
A(1,1) = 1;
b(1) = u_a;

% Solve linear system
U = A\b;

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



function [A_local, b_local] = ...
```

```
    CalculateContributionOnElement(x, k, p, q, r, f)

% Calculate element length
h = x(2*k+1) - x(2*k-1);

% 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 contributions to A and b to zero
A_local = zeros(3,3);
b_local = zeros(3,1);

% Loop over quadrature points
for m = 1:M
    X = QuadPoints(m);

    % Evaluate local basis functions and their derivatives
    phi_local = [2*(1-X)*(0.5-X), 4*X*(1-X), 2*X*(X-0.5)];
    phi_prime_local = [4*X-3, 4-8*X, 4*X-1];

    %Evaluate p, q, r, f at quadrature points
    p_val = p(x(2*k-1)+h*X);
    q_val = q(x(2*k-1)+h*X);
    r_val = r(x(2*k-1)+h*X);
    f_val = f(x(2*k-1)+h*X);

    % Increment entries of A_local and b_local
    % with suitably weighted function evaluations
    for i=1:3
        for j=1:3
            A_local(i,j) = A_local(i,j) + ...
                h*QuadWeights(m)*( ...
                p_val/h/h*phi_prime_local(i)* ...
                phi_prime_local(j) + ...
                q_val/h*phi_local(i)* ...
                phi_prime_local(j) + ...
                r_val*phi_local(i)*phi_local(j));
        end
        b_local(i) = b_local(i) + h*QuadWeights(m)* ...
            f_val*phi_local(i);
    end
end
```