```
function [x, y, lnods, U] = Chap8_CalculateExampleFem(Nx,Ny)

% Define functions
p = @(x,y) exp(x);
q1 = @(x,y) 3*x;
q2 = @(x,y) 2*y;
r = @(x,y) -6;
f = @(x,y) -2*exp(x)*(1+x+6*y);
gn1 = @(x,y) 6*exp(x);
gn2 = @(x,y) 2*exp(x);

% Generate data structures needed to specify the mesh
[x, y, lnods, dir_nodes, neu_edges] = GenerateMesh(Nx, Ny);

% Initialise A and b to zero
A = sparse((Nx+1)*(Ny+1), (Nx+1)*(Ny+1));
b = zeros((Nx+1)*(Ny+1), 1);

% Loop over elements, calculating local contributions, and
% incrementing global A and b
for k=1:2*Nx*Ny
    [A_local, b_local] = ...
        CalculateContributionOnElement(x, y, lnods, k, ...
        p, q1, q2, r, f);

    A(lnods(k,:), lnods(k,:)) = ...
        A(lnods(k,:), lnods(k,:)) + A_local;
    b(lnods(k,:)) = b(lnods(k,:)) + b_local;
end

% Loop over edges on y=1, calculating local contributions
% from Neumann boundary conditions, and adding into the
% global b
for k=1:Nx
    b_local = CalculateContributionOnNeumannEdge(x, y, ...
        neu_edges, k, gn1);

    b(neu_edges(k,:)) = b(neu_edges(k,:)) + b_local;
end

% Loop over edges on x=-1, calculating local contributions
% from Neumann boundary conditions, and adding into the
% global b
for k=Nx+1:Nx+Ny
    b_local = CalculateContributionOnNeumannEdge(x, y, ...
        neu_edges, k, gn2);

    b(neu_edges(k,:)) = b(neu_edges(k,:)) + b_local;
end

% Handle Dirichlet boundary conditions
A(dir_nodes,:) = 0;
for i=1:length(dir_nodes)
    A(dir_nodes(i),dir_nodes(i)) = 1;
end
b(dir_nodes) = x(dir_nodes).^2 + 2*y(dir_nodes).^3;

% Compute ILU factorisation of A
```

```
[Lower, Upper] = ilu(A);

% Use ILU preconditioned GMRES to solve linear system
U = gmres(A, b, 100, [], 1000, Lower, Upper);

% Plot solution
trisurf(lnods, x, y, U);
xlabel('x')
ylabel('y')
zlabel('U')
```

```
% Function to generate data structures required by the mesh
function [x, y, lnods, dir_nodes, neu_edges] = ...
    GenerateMesh(Nx, Ny)

% Set entries of lnods
lnods = zeros(2*Nx*Ny, 3);
for j=1:Ny;
    for i=1:Nx;
        ele = 2*(j-1)*Nx+2*i-1;
        lnods(ele,:) = ...
            [(j-1)*(Nx+1)+i, (j-1)*(Nx+1)+i+1, j*(Nx+1)+i];

        ele = ele+1;
        lnods(ele,:) = ...
            [j*(Nx+1)+i+1, j*(Nx+1)+i, (j-1)*(Nx+1)+i+1];
    end
end

% Set entries of x and y over the interval from 0 to 1
x = zeros((Nx+1)*(Ny+1),1);
y = zeros((Nx+1)*(Ny+1),1);

for i=1:Nx+1
    x(i:Nx+1:(Nx+1)*(Ny+1)) = 2*(i-1)/Nx-1;
end

for j=1:Ny+1
    y((j-1)*(Nx+1)+1:j*(Nx+1)) = 2*(j-1)/Ny-1;
end

% Specify locations of Dirichlet boundary conditions
% Nodes on y=-1
dir_nodes(1:Nx+1) = 1:Nx+1;
% Nodes on x=1
dir_nodes(Nx+2:Nx+Ny+1) = 2*(Nx+1):Nx+1:(Ny+1)*(Nx+1);

% Specify locations of Neumann boundary conditions
% Edges on y=1
neu_edges(1:Nx,:) = [((Ny+1)*(Nx+1):-1:Ny*(Nx+1)+2)', ...
    ((Ny+1)*(Nx+1)-1:-1:Ny*(Nx+1)+1)'];
% Edges on x=-1
neu_edges(Nx+1:2*Nx,:) = [(Ny*(Nx+1)+1:-(Nx+1):Nx+2)', ...
    ((Ny-1)*(Nx+1)+1:-(Nx+1):1)'];
```

```
% Calculate local contributions to A and b from element k
function [A_local, b_local] = ...
    CalculateContributionOnElement(x, y, lnods, k, p, ...
        q1, q2, r, f)

% Define x and y coordinate of each node
x1 = x(lnods(k,1));
x2 = x(lnods(k,2));
x3 = x(lnods(k,3));
y1 = y(lnods(k,1));
y2 = y(lnods(k,2));
y3 = y(lnods(k,3));

% Calculate Jacobian of transformation between canonical
% element and element k
detF = (x2-x1)*(y3-y1) - (x3-x1)*(y2-y1);

% Evaluate derivatives of basis functions in terms of (x,y)
% coordinates
Finv = [y3-y1, x1-x3; y1-y2, x2-x1]/detF;
dphi_dX = [-1, -1; 1, 0; 0, 1];
dphi_dx = dphi_dX*Finv;

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

% Loop over quadrature points to calculate local ...
% contributions to A and b
for n=1:M
    for m=1:M
        % Set values of basis functions
        phi(1) = 1-QuadPoints(m)- ...
            (1-QuadPoints(m))*QuadPoints(n);
        phi(2) = QuadPoints(m);
        phi(3) = (1-QuadPoints(m))*QuadPoints(n);

        % Set values of x and y
        x_val = x1*phi(1) + x2*phi(2) + x3*phi(3);
        y_val = y1*phi(1) + y2*phi(2) + y3*phi(3);

        % Evaluate functions of x and y
        p_val = p(x_val, y_val);
        q1_val = q1(x_val, y_val);
        q2_val = q2(x_val, y_val);
        r_val = r(x_val, y_val);
        f_val = f(x_val, y_val);

        % Add weighted quadrature value to local A and b
        for i=1:3
```

```
            for j=1:3
                A_local(i,j) = A_local(i,j) + ...
                    detF*QuadWeights(m)* ...
                    QuadWeights(n)*(1-QuadPoints(m))* ...
                    (p_val*(dphi_dx(i,1)*dphi_dx(j,1) + ...
                    dphi_dx(i,2)*dphi_dx(j,2)) + ...
                    q1_val*phi(i)*dphi_dx(j,1) + ...
                    q2_val*phi(i)*dphi_dx(j,2) + ...
                    r_val*phi(i)*phi(j));
            end
            b_local(i) = b_local(i) + ...
                detF*QuadWeights(m)*QuadWeights(n)* ...
                (1-QuadPoints(m))*f_val*phi(i);
        end
    end
end
```

```
function b_local = ...
    CalculateContributionOnNeumannEdge(x, y, neu_edges, ...
    k, gn)

% Define x and y coordinate of each node
x1 = x(neu_edges(k,1));
x2 = x(neu_edges(k,2));
y1 = y(neu_edges(k,1));
y2 = y(neu_edges(k,2));

% Calculate length of edge
h = sqrt((x1-x2)^2 + (y1-y2)^2);

% 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 b to zero
b_local = zeros(2,1);

% Loop over quadrature points to calculate local
% contributions to b
for m=1:M
    % Define basis functions at each quadrature point
    phi = [1-QuadPoints(m); QuadPoints(m)];

    % Define values of x and y
    x_val = x1*phi(1) + x2*phi(2);
    y_val = y1*phi(1) + y2*phi(2);

    % Evaluate g_N
    gn_val = gn(x_val, y_val);

    % Add weighted contribution to b
    b_local = b_local + h*QuadWeights(m)*gn_val*phi;
```

```
end
```