```
function [x, y, lnods, U] = Chap7_CalculateExampleFem(Nx,Ny)

% Generate data structures needed
[x, y, lnods, dir_nodes] = 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
% adding the local contributions into the global A and b
for k=1:2*Nx*Ny
    [A_local, b_local] = ...
        CalculateContributionOnElement(x, y, lnods, k);

    A(lnods(k,:), lnods(k,:)) = ...
        A(lnods(k,:), lnods(k,:)) + A_local;
    b(lnods(k,:)) = b(lnods(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) = 0;

% Compute ILU factorization of A
[Lower, Upper] = ilu(A);

% Use ILU preconditioned GMRES to solve linear system
U = gmres(A, b, 30, [], 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] = 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)) = (i-1)/Nx;
end

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

% Specify locations of Dirichlet boundary conditions
% Nodes on y=0
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);
% Nodes on y=1
dir_nodes(Nx+Ny+2:2*Nx+Ny+1) = ...
   (Ny+1)*(Nx+1)-1:-1:Ny*(Nx+1)+1;
% Nodes on x=0
dir_nodes(2*Nx+Ny+2:2*(Nx+Ny)) = ...
    (Ny-1)*(Nx+1)+1:-(Nx+1):Nx+2;



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

% 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;

% Assemble local contribution to A
A_local = zeros(3,3);
for i=1:3
    for j=1:3
        A_local(i,j) = detF*0.5* ...
            (dphi_dx(i,1)*dphi_dx(j,1) + ...
            dphi_dx(i,2)*dphi_dx(j,2));
    end
end

% Assemble local contribution to b
```

```
b_local = ones(3,1)*detF/6;
```