```
function prob4(N)

%NODES
nodes = zeros(((N+1)^2) , 3);
%nodes(k,1), nodes(k,2) are the x, y coordinates of the k^th node
%nodes(k,3) is 1 if the k^th node is free, or else -1 if the node is↙
constrained

[X, Y] = meshgrid((1:(N-1)),(1:(N-1)));
X=(X’)/N; Y=(Y’)/N; %for renormalizing into the unit square
nodes((1:((N-1)^2)),1) = reshape(X, ((N-1)^2), 1);
nodes((1:((N-1)^2)),2) = reshape(Y, ((N-1)^2), 1);
nodes((1:((N-1)^2)),3) = ones(((N-1)^2),1);

%bottom boundary points
    nodes((((N-1)^2+1):((N-1)^2+N)),1) = ([0:(N-1)]’)./N; %x coords
    nodes((((N-1)^2+1):((N-1)^2+N)),2) = zeros(N,1); %y coords
    nodes((((N-1)^2+1):((N-1)^2+N)),3) = -ones(N,1);
%right boundary nodes
    nodes((((N-1)^2+N+1):((N-1)^2+2*N)),1) = ones(N,1); %x coords
    nodes((((N-1)^2+N+1):((N-1)^2+2*N)),2) = ([0:(N-1)]’)./N; %y coords
    nodes((((N-1)^2+N+1):((N-1)^2+2*N)),3) = -ones(N,1);
%top boundary nodes
    nodes((((N-1)^2+2*N+1):((N-1)^2+3*N)),1) = flipud(([1:N]./N)’); %x↙
coords
    nodes((((N-1)^2+2*N+1):((N-1)^2+3*N)),2) = ones(N,1); %y coords
    nodes((((N-1)^2+2*N+1):((N-1)^2+3*N)),3) = -ones(N,1); %y coords
%left boundary nodes
    nodes((((N-1)^2+3*N+1):((N-1)^2+4*N)),1) = zeros(N,1); %x coords
    nodes((((N-1)^2+3*N+1):((N-1)^2+4*N)),2) = flipud(([1:N]./N)’); %y↙
coords
    nodes((((N-1)^2+3*N+1):((N-1)^2+4*N)),3) = -ones(N,1); %x coords

% TRIANGLES
triangles = zeros(2*N^2,3);
% triangle(k,i) records the global node number of the i^th local node of↙
triangle k

%auxilliary definitions
triangle_coords = zeros(2*N^2,6);
oddtriangle = [0,1/N, 0, 0, 1/N, 1/N]; %equal to triangle_coords(1,:)
eventriangle = [1/N, 0, 1/N, 1/N, 0, 0]; %equal to triangle_coords(2,:)
% provides coordinates for triangles
% e.g., triangle_coords(k,:) = [(x1,y1), (x2,y2), (x3,y3)], where (xi,yi)↙
are the coordinates for the i^th local node of triangle k

% every odd/even triangle given the same info as triangle 1/2, resp.
triangle_coords = (mod(1:2*N^2,2)’)*oddtriangle + ((1-mod(1:2*N^2,2))’)↙
*eventriangle;
% each odd triangle info shifted by appropriate amount - x and then y↙
coordinates
triangle_coords = triangle_coords + ((mod(1:2*N^2,2).*(mod((1:2*N^2)-1,2*N)↙
/2))’)*[1/N, 0, 1/N, 0, 1/N, 0];
triangle_coords = triangle_coords + ((mod(1:2*N^2,2).*(floor(((1:2*N^2)-1)/↙
(2*N))))’)*[0, 1/N, 0, 1/N, 0, 1/N];
% each even triangle info shifted by appropriate amount - x and then y↙
```

```
coordinates
triangle_coords = triangle_coords + (((1-mod(1:2*N^2,2)).*(mod((1:2*N^2)-2,2↙
*N)/2))’)*[1/N, 0, 1/N, 0, 1/N, 0];
triangle_coords = triangle_coords + (((1-mod(1:2*N^2,2)).*(floor(((1:2*N^2)↙
-1)/(2*N))))’)*[0, 1/N, 0, 1/N, 0, 1/N];

% adjust for special top left and bottom right corners
BRL = [triangle_coords(2*N-1,3:4), triangle_coords(2*N,1:2), triangle_coords↙
(2*N-1,1:2)];
BRR = [triangle_coords(2*N,3:4), triangle_coords(2*N-1,1:2), triangle_coords↙
(2*N,1:2)];
triangle_coords(2*N-1,:) = BRL;
triangle_coords(2*N,:) = BRR;
TLL = [triangle_coords(2*(N-1)*N+1,3:4), triangle_coords(2*(N-1)*N+2,1:2),↙
triangle_coords(2*(N-1)*N+1,1:2)];
TLR = [triangle_coords(2*(N-1)*N+2,3:4), triangle_coords(2*(N-1)*N+1,1:2),↙
triangle_coords(2*(N-1)*N+2,1:2)];
triangle_coords(2*(N-1)*N+1,:) = TLL;
triangle_coords(2*(N-1)*N+2,:) = TLR;

for k=1:(2*N^2)
    for l=1:3
        test=(abs(nodes(:,1)-triangle_coords(k,2*l-1))<eps).*(abs(nodes(:,2)↙
-triangle_coords(k,2*l))<eps);
        triangles(k,l)=(1:((N+1)^2))*test;
    end
end

%VISUALIZE MESH
for k=1:(2*N^2)
    plot(triangle_coords(k,[1, 3]), triangle_coords(k,[2, 4])); hold on;
    plot(triangle_coords(k,[3, 5]), triangle_coords(k,[4, 6])); hold on;
    plot(triangle_coords(k,[5, 1]), triangle_coords(k,[6, 2])); hold on;
end

nodes
triangles
```