A scalar field $\phi(x, y)$ is governed by the partial differential equation $-\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)-\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)=f(x,y)$ in the $2a\times 2b$ domain shown in the figure. The boundary conditions on the four sides of the domain are also shown. Derive the weak form of this problem using one of the techniques discussed in the class. Carefully note the kind of non-essential boundary conditions that can be applied for this problem.

Considering that Q4 elements will be used to discretise the problem, derive the element stiffness matrix and nodal body force vector.

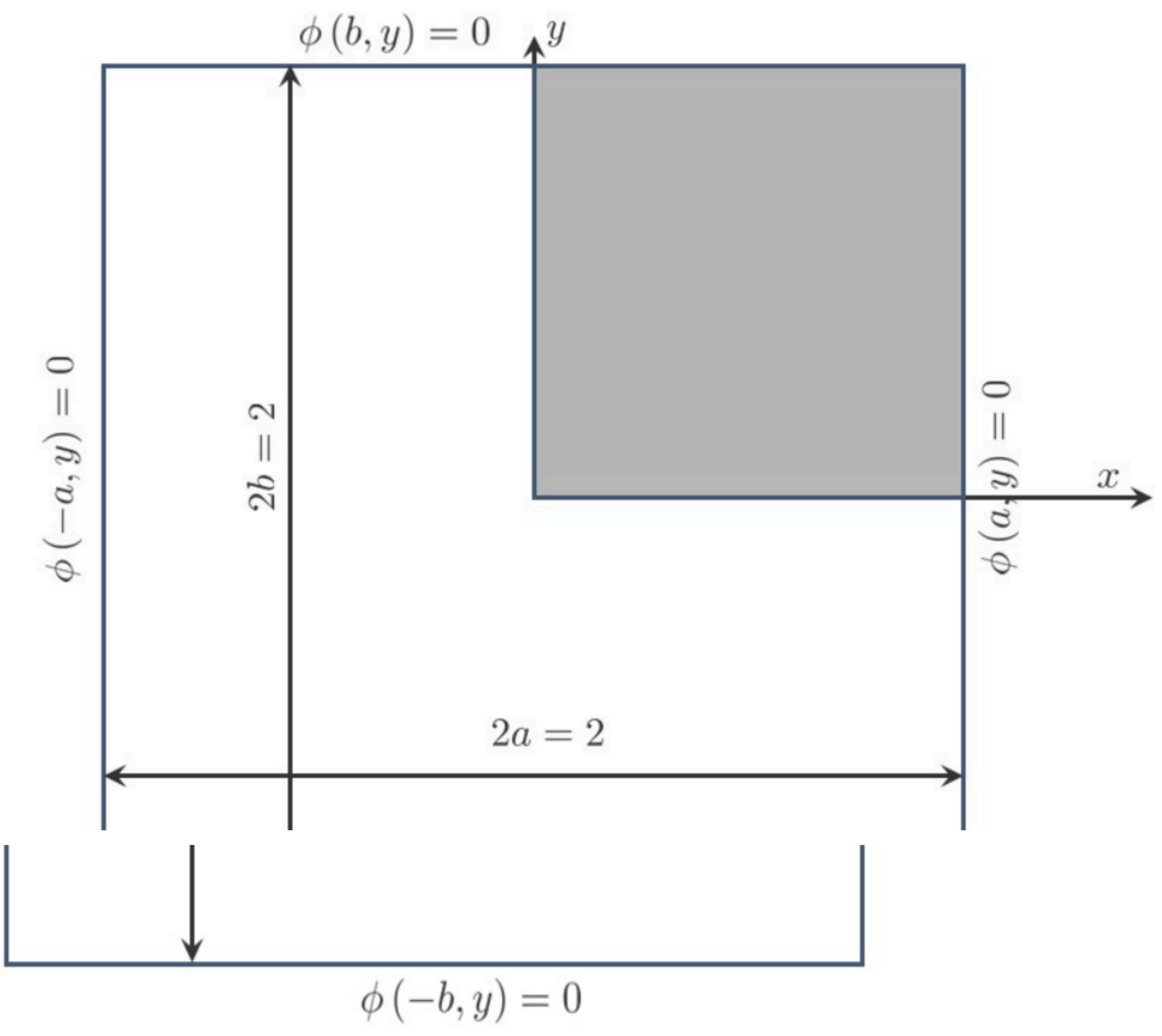
Now, because of the symmetry in the problem, it is decided to model only the shaded part. This shaded part is discretised into $N\times N$ Q4 elements. Appropriate boundary conditions have to applied to the four sides of the shaded domain. Apply these boundary conditions carefully and logically.

We will use $N=8, a=b=1$. Also $k=1$ and $f(x,y)=f_0=1$.

You can use a few useful functions that are provided. A function `genelem(L,N)` outputs the nodal coordinates in the vectors `x` and `y` and the connectivity array in `conn`. It should also display the mesh you have created.

Another function `shape_functionsQ4.m` guides you through the steps of forming the Q4 shape functions, Jacobian matrix `J` and arrays containing $\left\langle \frac{dN_1}{dx}\ \frac{dN_2}{dx}\ \frac{dN_3}{dx}\ \frac{dN_4}{dx}\right\rangle$ and $\left\langle \frac{dN_1}{dy}\ \frac{dN_2}{dy}\ \frac{dN_3}{dy}\ \frac{dN_4}{dy}\right\rangle$. You should complete this first before you start with the main code.

In the main code, you need to generate the mesh, form the elemental stiffness matrix and force vectors, assemble them and use the boundary conditions **for the shaded domain** in order to derive the solution for $\mathbf{\Phi}$, the vector of nodal values of $\phi(x,y)$.



```matlab
% Q4 element problem
L = 1; N= 8;
[X,Y,conn] = genelem(L,N);
nnod=length(X);
nelem=size(conn,1);
% initialise global stiffness and force vector
Kg = zeros(nnod,nnod);
Fg = zeros(nnod,1);
% material properties
k = 1;
f0 = 1;
% create destination array
for ielem = 1:nelem

    for ielem = 1:nelem
   dest = conn(ielem,:);
end
end
% for Q4 element we will perform Gauss Quadrature of sufficient order
% 1-d sampling points

% 2-d sampling points in r and s arrays
r3 = [-1/sqrt(3)  1/sqrt(3)];   % 2-point Gauss in 1D

% 2-d sampling points in r and s arrays (tensor product)
[rr,ss] = meshgrid(r3,r3);
r = rr(:)';
s = ss(:)';
% corresponding weights calculated as w(i)*w(j) as explained in class
weights=ones(1,length(r));;
% form element stiffness and force vector
for ielem = 1:nelem
% get nodal coordinates for ielem
 nodes = conn(ielem,:);
   x = X(nodes)';
   y = Y(nodes)';
   % initialise Ke and Fe
   Ke = zeros(4,4);
   Fe = zeros(4,1);
% initialise Ke and Fe
   %Ke=zeros();
   %Fe=zeros(,1);
```

```
        % loop over gauss points
    for lint = 1:length(weights)
        [Nshape,dNdx,dNdy,J] = shape_functionsQ4(x,y,r(lint),s(lint));
        jacob = det(J);
        wt = weights(lint);
        % stiffness matrix (diffusion)
        % Ke_ij = k * \int (dNi/dx * dNj/dx + dNi/dy * dNj/dy) dOmega
        Ke = Ke + k*( (dNdx')*dNdx + (dNdy')*dNdy ) * jacob * wt;
        % force vector Fe = \int N^T * f0 dOmega
        Fe = Fe + (Nshape') * f0 * jacob * wt;
end  % gauss loop ends
% assemble

   %Fg(dest(ielem,idof)) = ;

   %Kg(dest(ielem,idof),dest(ielem,jdof)) = ;

   % assemble
 Fg(nodes) = Fg(nodes) + Fe;
 Kg(nodes,nodes) = Kg(nodes,nodes) + Ke;

end % element loop ends
% boundary conditions
tol = 1e-12;
%bound = find( abs(X-1) < tol | abs(Y-1) < tol );
% dof array containing dofs that are fixed
bound =find( abs(X-1) < tol | abs(Y-1) < tol ); ;
Kprime = Kg;
Fprime = Fg;
% apply fixed boundary conditions
for ii=1:length(bound)
   bd = bound(ii);
   Kprime(bd,:) = 0;
   Kprime(:,bd) = 0;
   Kprime(bd,bd) = 1;
   Fprime(bd) = 0;
end
phi = inv(Kprime)*Fprime;

testvar1 = phi(3);
testvar2 = phi(41);
% ============================================================
% function for generating mesh. nothing to be done here
%=============================================================
function[x,y,conn] = genelem(L,N)
% function to generate the mesh for the problem
% elements along x
nx=N;
%elements along y
ny=N;
Lx=L;
Ly=L;
nelem = nx*ny;
nnode = (nx+1)*(ny+1);
delx = Lx/nx;
dely = Ly/ny;
% generate nodal coordinates
for inode = 1:nnode
   nrow = ceil(inode/(nx+1)) - 1;
   y(inode) = nrow*dely;
   ncol = inode - (nx+1)*nrow;
   x(inode) = (ncol - 1)*delx;
end
% generate connectivity
for ielem = 1:nelem
   nerow= ceil(ielem/nx);
   necol = ielem - (nerow - 1)*nx ;
   node1 = (nerow - 1)*(nx + 1) + necol;
   node2 = node1 + 1;
   node4 = node1 + nx + 1;
   node3 = node4+1;
   conn(ielem,:) = [node1 node2 node3 node4];
end
% draw the mesh to verify everything is fine
for ii = 1:nelem
   n1 = conn(ii,1); x1 = x(n1); y1 = y(n1);
   n2 = conn(ii,2); x2 = x(n2); y2 = y(n2);
   n3 = conn(ii,3); x3 = x(n3); y3 = y(n3);
   n4 = conn(ii,4); x4 = x(n4); y4 = y(n4);
   plot([x1 x2 x3 x4 x1],[y1 y2 y3 y4 y1],'k');
   hold on;
   x0=0.25*(x1+x2+x3+x4); y0 = 0.25*(y1+y2+y3+y4);
   text(x0,y0,num2str(ii));
end
end

% =====================================================
% shape function routine for Q4 elements
% =======================================================
function[N,dNdx,dNdy,J] = shape_functionsQ4(x,y,r,s)
% x and y are 4X1 vectors that store the X and Y nodal coordinates of
the element
% whose shape functions are being calculated. r, s are scalars
containing the (r,s) point
% in the parent element at which the shape functions are to be returned.
xg=r;yg=s;
% shape functions
N = [ 0.25*(1-r)*(1-s), ...
     0.25*(1+r)*(1-s), ...
     0.25*(1+r)*(1+s), ...
     0.25*(1-r)*(1+s) ];

% shape function r and s derivatives in 1X4 arrays
dNdr = [ -0.25*(1-s),  0.25*(1-s),  0.25*(1+s), -0.25*(1+s) ];
dNds = [ -0.25*(1-r), -0.25*(1+r),  0.25*(1+r),  0.25*(1-r) ];

% jacobian matrix
J = [ dNdr * x,  dNdr * y;
     dNds * x,  dNds * y ];
% shape function x and y derivatives to be formed in 1X4 arrays
Gamma = inv(J);
% shape function x and y derivatives (1x4)
dNdx = Gamma(1,1)*dNdr + Gamma(1,2)*dNds;
dNdy = Gamma(2,1)*dNdr + Gamma(2,2)*dNds;
end
```

```
% x and y are 4X1 vectors that store the X and Y nodal coordinates of
the element
% whose shape functions are being calculated. r, s are scalars
containing the (r,s) point
% in the parent element at which the shape functions are to be returned.
```