## FEM2D

This is the main driver function which is called to run the program. It has the input and output file as its two inputs.

The notations for the input quantities are the same as defined in the input file.

**K** and **Q** are the Global Stiffness and Right Side Vectors. **K_cap** and **Q_cap** are the modified Stiffness and Right side vectors after applying the essential boundary condition.**u** and **v** are the x and y displacements respectively.

```
function FEM2D(input_file,output_file)
    %PreProcessing
    load(input_file);
    %Processing
    [K,Q] = computeKQ(p_e,p_b,n_b,n_e,n_n,lb,C,C_b,t_star,S);
    [K_cap,Q_cap] = ApplyEBC(K,Q,Bu,n_n);
    U = linsolve(K_cap,Q_cap);
    %PostProcessing
    u = U(1:2:end);
    v = U(2:2:end);
    output = [X,u,v];
    writematrix(output,output_file);
end
```

## computeKQ

Computing the Global Coefficient matrix and the Global right side vector

```
function [K,Q] = computeKQ(p_e,p_b,n_b,n_e,n_n,lb,C,C_b,t_star,S)
    K = zeros(2*n_n,2*n_n); %initialising the K matrix with all zeroes
    Q = zeros(2*n_n,1); %initialising the Q vector with all zeroes
    w = [1, 1]; %inputing the weights
    zhi = [-0.577350269189626, 0.577350269189626]; %inputing the corresponding zhi
    for e = 1:n_e
        ke = computeElementalStiffness(p_e,w,zhi,S); %computing elemental stiffness
using the pre-written function
        Ke = GlobalAssemblyStiffness(p_e,n_n,ke,e,C); %global assembly of the
coefficient matrix
        K = K + Ke;
    end
    for b = 1:n_b
        qb = computeTraction(p_b,w,zhi,lb,t_star); %computing the elemental right
side vector using the pre-written function
        Qb = GlobalAssemblyTraction(p_b,n_n,C_b,b,qb); %global assembly of the
right side vector
        Q = Q + Qb;
    end
end
```

**computeElementalStiffness**

Computing the Element Coefficient Matrix

```
function ke = computeElementalStiffness(p_e,w,zhi,S)
    ke = zeros(2*p_e,2*p_e); %Creating the matrix
    ng = length(w); % number of gauss points
    for i = 1:ng
        for j = 1:ng
            Be = computeElementShapeFuncDeriv(zhi(i),zhi(j)); % computing
derivative of shape function vector
            ke = ke + w(i)*w(j)*Be'*S*Be; % using the gauss legendre Integration
scheme
        end
    end
end
```

**comuteElementShapeFuncDeriv**

This function Computes the elemental Shape function vector at given xxi and eeta

```
function [B] = computeElementShapeFuncDeriv(xxi, eeta)
    syms s;
    L1 = (1/2)*(1-s);
    Lp1 = (-1/2);
    L2 = (1/2)*(1+s);
    Lp2 = (1/2);
    B = zeros(2, 8);
    B(1, 1) = (Lp1)*subs(L1, s, eeta);
    B(1, 3) = (Lp2)*subs(L1, s, eeta);
    B(1, 5) = (Lp2)*subs(L2, s, eeta);
    B(1, 7) = (Lp1)*subs(L2, s, eeta);
    B(2, 2) = subs(L1, s, xxi)*(Lp1);
    B(2, 4) = subs(L2, s, xxi)*(Lp1);
    B(2, 6) = subs(L2, s, xxi)*(Lp2);
    B(2, 8) = subs(L1, s, xxi)*(Lp2);
    B(3, 1) = subs(L1, s, xxi)*(Lp1);
    B(3, 2) = (Lp1)*subs(L1, s, eeta);
    B(3, 3) = subs(L2, s, xxi)*(Lp1);
    B(3, 4) = (Lp2)*subs(L1, s, eeta);
    B(3, 5) = subs(L2, s, xxi)*(Lp2);
    B(3, 6) = (Lp2)*subs(L2, s, eeta);
    B(3, 7) = subs(L1, s, xxi)*(Lp2);
    B(3, 8) = (Lp1)*subs(L2, s, eeta);

end
```

**GlobalAssemblyStiffness**

Applying the simplified Assembly relations for the Coefficient Matrix

```
function Ke = GlobalAssemblyStiffness(p_e,n_n,ke,e,C)
    Ke = zeros(2*n_n,2*n_n); %initialising the global K matrix to be zero
    for p = 1:p_e
        for q = 1:p_e
            r = C(e,p); % r is the global node number of the pth local node of eth
domain element
            s = C(e,q); % s is the global node number of the qth local node of eth
domain element
            %Using the simplified assembly relations for the coefficient matrix
            Ke(2*r-1,2*s-1) = ke(2*p-1,2*q-1);
            Ke(2*r,2*s-1) = ke(2*p,2*q-1);
            Ke(2*r-1,2*s) = ke(2*p-1,2*q);
            Ke(2*r,2*s) = ke(2*p,2*q);
        end
    end
end
```

**computeTraction**

Applying the gauss legendre integration Scheme to compute the boundary elemental right side vector

```
function qb = computeTraction(p_b,w,zhi,lb,t_star)
    qb = zeros(2*p_b,1); %initialising the right side vector to be zero
    ng = length(w); %number of gauss points
    for k = 1:ng
        Nb = computeBoundaryShapeFunc(zhi(k));
        qb = qb + (lb/2)*w(k)*Nb'*t_star; %using the gauss legendre integration
scheme
    end
end
```

**computeBoundaryShapeFunc**

Computing the Element Boundary Shape function

```
function [Nb] = computeBoundaryShapeFunc(xxi)
    Nb = zeros(2, 4);
    Nb(1, 1) = (1/2)*(1-xxi);
    Nb(1, 3) = (1/2)*(1-xxi);
    Nb(2, 2) = (1/2)*(1+xxi);
    Nb(2, 4) = (1/2)*(1+xxi);
end
```

**GlobalAssemblyTraction**

Applying the simplified Assembly relations for the right side vector

```
function Qb = GlobalAssemblyTraction(p_b,n_n,C_b,b,qb)
    Qb = zeros(2*n_n,1); %initialing the global right side vector with zero
    for p = 1:p_b
        r = C_b(b,p); % r is the global node number of the pth local node of bth
boundary element
        %using the simplified assembly relations for the right side vector
        Qb(2*r-1) = qb(2*p-1);
        Qb(2*r) = qb(2*p);
    end
end
```

**ApplyEBC**

Applies the essential boundary condition that is specified over the C1 boundary and returns the modified Global Stiffness and Right Side vector

```
function [K,Q] = ApplyEBC(K,Q,Bu,n_n)
    % incorporating Essential Boundary Conditions
    i=1;
    % looping over number of points for which Essential Boundary Conditions is
specified
    while i<=length(Bu)
        index = Bu(i,1);
        u_index = Bu(i,2);
        v_index = Bu(i,3);
        j=1;
        while j<=2*n_n
            K(2*index-1,j) = 0;
            Q(j,1) = Q(j,1)-K(j,2*index-1)*u_index;
            K(2*index,j) = 0;
            Q(j,1) = Q(j,1)-K(j,2*index)*v_index;
            K(j,2*index-1) = 0;
            K(j, 2*index) = 0;
            j=j+1;
        end
        K(2*index-1, 2*index-1) = 1;
        K(2*index, 2*index) = 1;
        Q(2*index-1) = u_index;
        Q(2*index) = v_index;
        i=i+1;
    end
end
```