Table of Contents


```
classdef Triangle < StdRegions.TriangleBasic

    % Stdandard Element Triangle
    % Triangle properties(Inherit):
    %   nDim         - the dimension of element
    %   nFace        - the number of edges
    %   nNode        - the number of nodes
    %   nOrder       - the order of degrees
    %   nVertice - the number of vertices
    %   sName        - the name of element
    %   M            - Mass matrix
    %   invM         - inverse of Mass Matrix
    %   Dr           - Derivative Matrix of r
    %   Ds           - Derivative Matrix of s
    %   Drw          - Derivative Matrix of r in weak form
    %   Dsw          - Derivative Matrix of s in weak form
    %   r            - point coordinate of dim 1, [nNode x 1]
    %   s            - point coordinate of dim 2, [nNode x 1]
    % Triangle properties:
    %   nFaceNode           - nFace x nNode (at face element)
    %   Mes                 - face integral mass matrix of face nodes
    %   Mef                 - face integral mass matrix of all nodes
    % Triangle methods:
    %   getNodeListAtFace        - return the number of nodes & node list
    %   getFaceListAtFace        - return the face list at spicific face
    %   getReorderFaceListAtFace - return the reorder face list of the spicific fa
    %   getFaceListToNodeList    - return the node list of face node
    %   getFaceGeometric         - return face normal vector & surface jacobi fact
    %   getEleGeometric          - return node Coordinate & dr/dx & jacobi factor
    % Usages:
    %   obj = Triangle(nOrder)
```

properties public

```
    properties
        nFaceNode           % nFace x nNode (at face element)
        Mes                 % face integral mass matrix of face nodes
        Mef                 % face integral mass matrix of all nodes
    end% properties
```

properties private

```
    properties(SetAccess=private, GetAccess=private)
```

```
        nPerBoundaryNode   % [nFace x 1] number of nodes at every boundary
    end% properties private
    methods
```

function Triangle

```
        function obj = Triangle(nOrder)
            obj = obj@StdRegions.TriangleBasic(nOrder);
            obj.sName = 'Triangle';
            % face element
            LineFaceShape = StdRegions.LineBasic(nOrder);
            % 3 faces of line type
            obj.nPerBoundaryNode = [LineFaceShape.nNode; LineFaceShape.nNode; ...
                LineFaceShape.nNode];
            % nFaceNode = nFace x nNode
            obj.nFaceNode = sum(obj.nPerBoundaryNode);
            obj.Mes = getSmallFaceMassMatrix(obj, LineFaceShape);
            obj.Mef = getFullFaceMassMatrix(obj);
        end% func
```

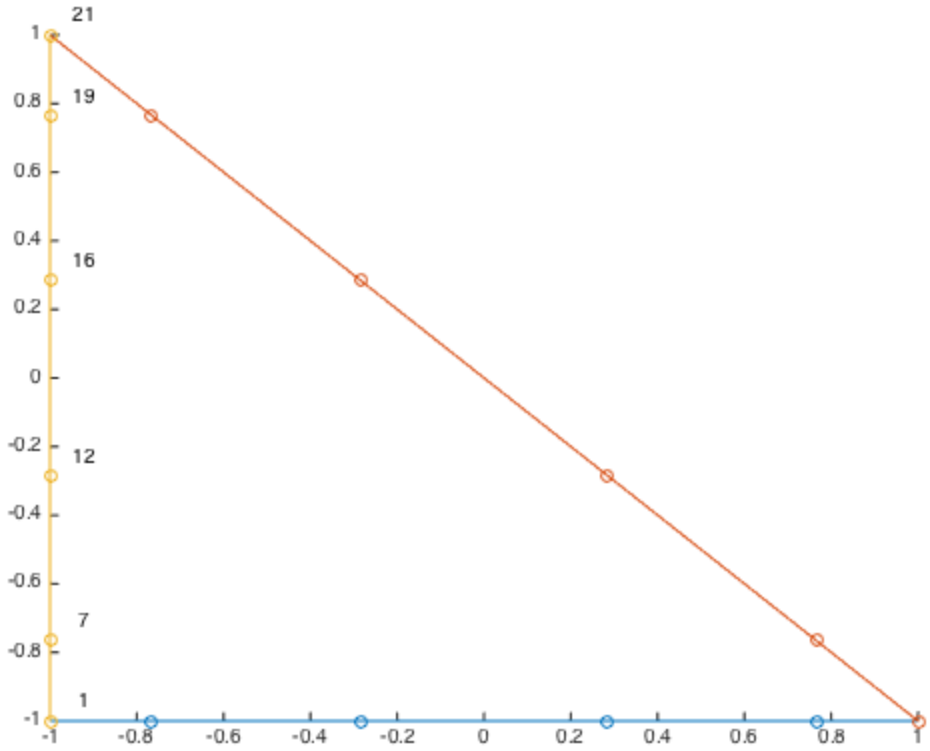
function getNodeListAtFace

return the num of node & node indicator at spicific face

```
tri = StdRegions.Triangle(5);
[n, temp] = tri.getNodeListAtFace(3)

n =
   6

temp =
    21
    19
    16
    12
     7
     1
```




```
function [n, nodelist] = getNodeListAtFace(obj,iface)
    facelist = obj.getFaceListAtFace(iface);
    faceListToNodeList = getFaceListToNodeList(obj);
    nodelist = faceListToNodeList(facelist);
    n = obj.nPerBoundaryNode(iface);
end% function

function facelist = getFaceListAtFace(obj,iface)
    % return the face list at spicific face
    if (iface ==1 )
        contour = 0;
    else
        contour = sum( obj.nPerBoundaryNode(1:iface-1) );
    end
    facelist = contour+1:contour+obj.nPerBoundaryNode(iface);
end% function

function facelist = getReorderFaceListAtFace(obj, iface, vorder)
    % return the reorder face list of the spicific face
    % with the vertice order reformed
    facelist = getFaceListAtFace(obj, iface);
    localList = StdRegions.Line.reorderLineNodeList(obj.nOrder, vorder);
    facelist(:) = facelist(localList);
end% function

function FacelistToNodelist = getFaceListToNodeList(obj)
```

```
            % return the node list of face node counterclockwise
%             MAXERROR = 10^-10;
            perfaceNode = obj.nFaceNode/3;   % No. of node on sigle face
            FacelistToNodelist = zeros(obj.nFaceNode,1);
            array = perfaceNode:-1:1;
            % face 1, s = -1
            FacelistToNodelist(1:perfaceNode) = 1:perfaceNode;
%             FacelistToNodelist(1:perfaceNode) = ...
%                 find(abs(obj.s - -1)< MAXERROR);
            % face 3, r = -1
            temp = ones(perfaceNode, 1);
            for irow = 2:perfaceNode
                temp(irow) = temp(irow - 1) + array(irow-1);
            end
            FacelistToNodelist(3*perfaceNode:-1:2*perfaceNode+1) = temp;
%             FacelistToNodelist(2*perfaceNode+1:3*perfaceNode) = ...
%                 find(abs(obj.r - -1)< MAXERROR);
            % face 2, r+s = 0
            temp2 = perfaceNode*ones(perfaceNode, 1);
            temp2(2:end-1) = temp(3:end) -1;
            temp2(end) = temp(end);
            FacelistToNodelist(perfaceNode+1:2*perfaceNode) = temp2;
%             FacelistToNodelist(perfaceNode+1:2*perfaceNode) = ...
%                 find(abs(obj.r + obj.s)< MAXERROR);

        end% function

        function [nx, ny, sJ] = getFaceGeometric(obj, x, y)
            % get Face Normal vector & surface jacobi factor
            % Input:    x - node coordinate, size [nNode, nElement]
            %           y - node coordinate, size [nNode, nElement]
            % Output:   nx - outward vector
            %           ny - outward vector
            %           sJ - face jacobi factor
            K = size(x, 2);
            xr = obj.Dr*x; yr = obj.Dr*y; xs = obj.Ds*x; ys = obj.Ds*y;
            % interpolate geometric factors to face nodes
            Fmask = obj.getFaceListToNodeList;
            fxr = xr(Fmask, :); fxs = xs(Fmask, :);
            fyr = yr(Fmask, :); fys = ys(Fmask, :);
            % build normals
            Nfp = obj.nFaceNode/3;
            nx = zeros(3*Nfp, K); ny = zeros(3*Nfp, K);
            fid1 = (1:Nfp)'; fid2 = (Nfp+1:2*Nfp)'; fid3 = (2*Nfp+1:3*Nfp)';
            % face 1
            nx(fid1, :) =  fyr(fid1, :); ny(fid1, :) = -fxr(fid1, :);
            % face 2
            nx(fid2, :) =  fys(fid2, :)-fyr(fid2, :); ny(fid2, :) = -fxs(fid2, :)+
            % face 3
            nx(fid3, :) = -fys(fid3, :); ny(fid3, :) =  fxs(fid3, :);
            % normalise
            sJ = sqrt(nx.*nx+ny.*ny); nx = nx./sJ; ny = ny./sJ;
        end
```

```
        function [x, y, rx, sx, ry, sy, J] = getEleGeometric(obj, VX, VY)
            % get element Geometric Factor
            % Input:    vx - Vertic Coordinate, size [3(nVertice) x nElement]
            %           vy - Vertic Coordinate, size [3(nVertice) x nElement]
            % Output:   x  - node coordinate
            %           rx - dr/dx at nodes
            %           J  - jacobi factor
            assert((size(VX,1)==3 && size(VY,1)==3), 'transferToPhysic: input vx f
            x = 0.5*(-(obj.r+obj.s)*VX(1,:)+(1+obj.r)*VX(2,:)+(1+obj.s)*VX(3,:));
            y = 0.5*(-(obj.r+obj.s)*VY(1,:)+(1+obj.r)*VY(2,:)+(1+obj.s)*VY(3,:));
            xr = obj.Dr*x; xs = obj.Ds*x; yr = obj.Dr*y; ys = obj.Ds*y; J = -xs.*y
            rx = ys./J; sx =-yr./J; ry =-xs./J; sy = xr./J;
        end

    end% methods public

    methods(Hidden)
        function FaceMassMatrixSmall = getSmallFaceMassMatrix(obj, LineShape)
            % getSmallFaceMassMatrix
            % $M^f_{i,j} = \int_{\partial \Omega}l_i l_{fj} ds$,
            % size [nNode x nFaceNode]

            % allocate the Face Mass Matrix
            FaceMassMatrixSmall = zeros(obj.nNode,obj.nFaceNode);
            for ib = 1:obj.nFace
                % iFaceList: the No. of ibth boundary local face node list.
                iFaceList = obj.getFaceListAtFace(ib);
                [~,iFaceNodeList] = obj.getNodeListAtFace(ib);
                FaceMassMatrixSmall(iFaceNodeList,iFaceList) = LineShape.M;
            end
        end %function getSmallFaceMassMatrix


        function FaceMassMatrixFull = getFullFaceMassMatrix(obj)
            % getFullFaceMassMatrix
            % $M^f_{i,j} = \int_{\partial \Omega}l_i l_j ds$, size [nNode x nNode]

            % allocate the Face Mass Matrix
            FaceMassMatrixFull = zeros(obj.nNode, obj.nNode);
            for ib = 1:obj.nFace
                [~,nodelist] = obj.getNodeListAtFace(ib);
                facelist = obj.getFaceListAtFace(ib);
                FaceMassMatrixFull(nodelist, nodelist) ...
                    = FaceMassMatrixFull(nodelist, nodelist) + ...
                    obj.Mes(nodelist,facelist);
            end
        end %function getFullFaceMassMatrix
    end% methods

end% classdef
```

Published with MATLAB® R2014b