Chapter 3, 2D Finite Element Spaces:

In this section, we will extend the framework for developing/studying finite element spaces to two dimensional partial differentia equations. Recall that this framework consists of the following main components:

- The mesh for a given domain Ω.
- The local finite element space on each element.
- The global finite element space defined on a mesh of the solutions Ω.
- Approximating functions by finite element functions on Ω. Approximations by both interpolation and L^2 projection will be discussed.

We will discuss the following related computations:

- The evaluation of a finite element function.
- Numerical quadratures for two dimensional integrals.
- Matrix and vector assemblers.

3.1, Meshes for 2D domains

Let $\Omega \subset \mathbf{R}^2$ be a polygonal domain. A mesh/triangulation $\mathcal{T}_h = \{K_i\}_{i=1}^{|\mathcal{T}_h|}$ is a collection of convex polygons (for examples, 2D simplexes or 2D quadrilaterals) called elements inside Ω such that

- $\overline{\Omega} = \cup_{K \in \mathcal{T}_h} K$.
- For every two element $K_1, K_2 \in \mathcal{T}_h$, $K_1 \cap K_2$ is either a common edge or a common vertex, or empty.

When $\Omega \subset \mathbf{R}^2$ has a curved boundary, we generally assume $\overline{\Omega} \approx \cup_{K \in \mathcal{T}_h} K$.

Again, in general, a mesh for 2D problems is characterized by 3 sets:

- The set of nodes: $\mathcal{N}_h = \{X_i\}_{i=1}^{|\mathcal{N}_h|}$ is the set of all the vertices of elements where $X_i = (x_i, y_i)$.
- The set of elements: $\mathcal{T}_h = \{K_i\}_{i=1}^{|\mathcal{T}_h|}$ in which each element is a polygon formed with some mesh nodes.
- The set of edges: $\mathcal{E}_h = \{e_i\}_{i=1}^{|\mathcal{E}_h|}$ is the set of all the edges of elements in the mesh.

For an element $K \in \mathcal{T}_h$, we let h_K be the diameter or the length of the longest edge of K, and the mesh size of $\mathcal{T}_h$ is defined as

$$h = \max_{K \in \mathcal{T}_h} h_K$$

Shape regular meshes: For each element K, we let d_K be the diameter of the maximum circle contained inside K. We call

$$c_K = \frac{h_K}{d_K} \tag{1}$$

the chunkiness parameter of K.

A family of meshes $\{\mathcal{T}_h\}_{h\geq 0}$ is shape regular provided that there exists a constant c_0 such that

$$c_K \leq c_0, \quad \forall K \in \mathcal{T}_h, \quad \forall \mathcal{T}_h \in \{\mathcal{T}_h\}_{h\geq 0} \tag{2}$$

Remarks: Shape regular is about the mesh quality for a family of meshes $\{\mathcal{T}_h\}_{h\geq 0}$ that guarantees the angles of the elements will not become too wide or too narrow when the meshes become finer.

Quasi-uniform meshes: A family of meshes $\{\mathcal{T}_h\}_{h\geq 0}$ is quasi-uniform provided that there exists a constant $\rho > 0$ such that

$$\rho < \frac{h_K}{h_{K'}} < \rho^{-1}, \ \forall K, K' \in \mathcal{T}_h, \ \forall \mathcal{T}_h \in \{\mathcal{T}_h\}_{h\geq 0}$$

Data structures for defining a mesh on a computer: As before, we use three arrays p, t and e to represent fundamental features of a mesh in a computer. In this course, we will use arrays p and t more often.

Array p is the node coordinate matrix/array. We index all the nodes (vertices of elements) in the mesh and put the x-y coordinates of the i-th node in the i-th column of p. Hence p is a $2 \times |\mathcal{N}_h|$ array.

Array t is the connectivity matrix/array associating the vertices of each element to some nodes. To define t, we index elements in $\mathcal{T}_h$, then design a data structure t such that the nodes/vertices in the k-th element can be efficiently retrieved. For a mesh formed by dissimilar elements, advanced data structures such as cell-arrays in Matlab can be used to implement t. The usual practice is to use a mesh formed by elements with the same geometry, such as triangles or quadrilaterals for a two dimensional domain Ω and t can implemented with a matrix in this case.

For a triangular mesh, t is a $3 \times |\mathcal{T}_h|$ array, its k-th column contains the indices of vertices of the k-th element arranged in a counter-clock order.
For a rectangular/quadrilaterial mesh, t is a $4 \times |\mathcal{T}_h|$ array, its k-th column contains the indices of vertices of the k-th element arranged in a counter-clock order or lexicographical order.

As before, we can pack both p and t into the structure `mesh`.

3.1.1, Generating meshes by PDE toolbox: Constructing satisfactory meshes efficiently for general domains in higher dimensions is a complicated task, especially for domains with complex geometries. On the other hand, especially for the purpose of studying finite element methods, there are many programs available for conveniently generating meshes for common 2D domains.

We will use basic mesh generating functions from Matlab's PDEtool to demonstrate the main ideas in mesh generation. These PDEtool functions are in three categories.

- Functions such as `decsg` is for defining the geometry of a domain.
- Functions such as `initmesh` and `refinemesh` are for constructing a mesh with a given geometry. `poimesh` is for generating a regular mesh on a rectangular domain.
- Functions `pdegplot` and `pdemesh` are for visualizing a domain and a mesh, respectively.

Remark: The e array from PDEtool's `initmesh`, `refinemesh`, and `poimesh` does not provide information rich enough for all the edges in a mesh.

Remark: The mesh generator in PDEtool is based on the well known Delaunay algorithm.

Function `decsg` uses a Geometry description matrix to generate a Decomposed Geometry array for mesh construction. PDEtool provides 4 basic 2D solids whose geometry description matrix is a 1D column array. Boolean operations on these basic 2D solids are allowed to form domains with more sophisticated geometry.

The geometry description matrix for these basic solids are as follows.

(1) For the circle domain: the 1st entry of this array should be 1. The second and third entries are for the x- and y-coordinates of the center. The 4th entry is the radius of the circle.

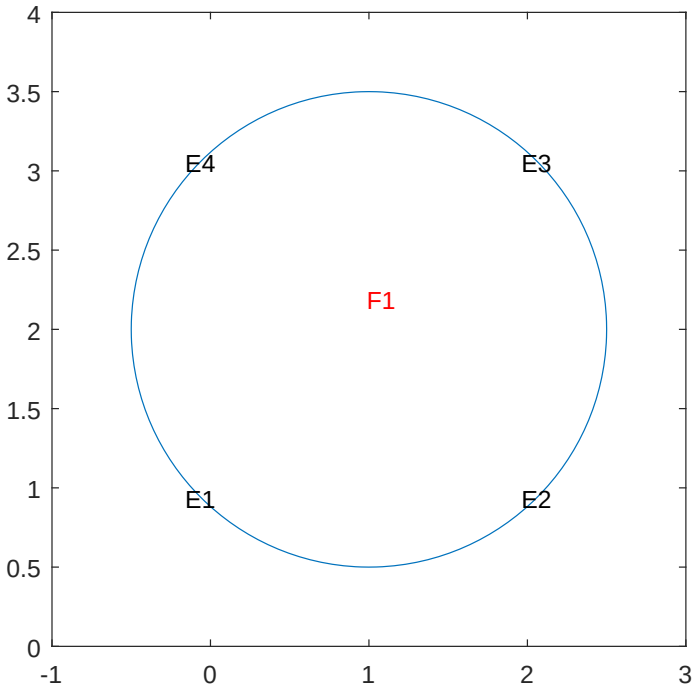
(2) For a polygon domain: the 1st entry of this array should be 2. The 2nd entry is an integer N for the number of line segments forming the boundary of the polygon. The following N entries contain the x-coordinates of the starting points of the line segments, and the following N entries are the y-coordinates of the starting points.

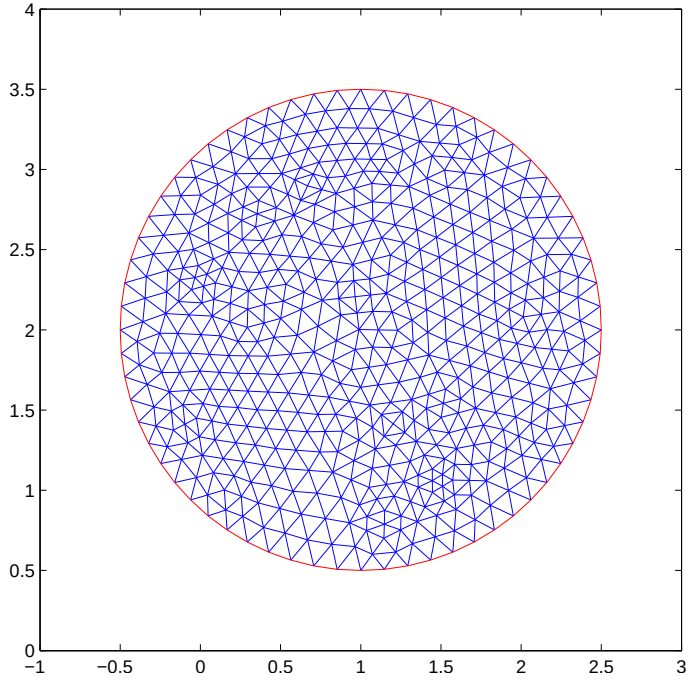
(3) For a rectangle domain: the 1st entry of this array should be 3, and the other entries are the same as those for the polygon domain.

(4) For an ellipse domain: the 1st entry of this array should be 4. The second and third entries are for the x- and y-coordinates of the center. The 4th and 5th entries are the major and minor axes of the ellipse. The 6th entry is the rotation angle (in radian) of the ellipse.

Here is a script to generate a mesh in a circular disk of radius 1.5 centered at $(1, 2)$.

```
C1 = [1,1,2,1.5]'; gd = decsg(C1); % define the geometry
figure(1); clf; % for displaying the domain
pdegplot(gd, 'EdgeLabels', 'on', 'FaceLabels', 'on');
axis equal; axis([-1, 3, 0, 4]); % adjust the plot
[p, e, t] = initmesh(gd); [p, e, t] = refinemesh(gd, p, e, t);
figure(2); clf; pdemesh(p, e, t);
axis equal; axis([-1, 3, 0, 4]); % display a mesh
mesh.p = p; mesh.t = t(1:3,:); % make a mesh data structure
```





We can also use Matlab's `triplot` command to plot a mesh

```
% define the domain
C1 = [1,1,2,1.5]'; gd = decsg(C1);
% make a mesh
[p, e, t] = initmesh(gd); [p, e, t] = refinemesh(gd, p, e, t);
mesh.p = p; mesh.t = t(1:3,:);
figure(3); clf; % for plotting the mesh
triplot(mesh.t', mesh.p(1, :)', mesh.p(2, :)');
axis equal; axis([-1, 3, 0, 4]);
```

Remark: Once the data structure `mesh` is constructed for a mesh, the x-y coordinates of the vertices in the k-th element in this mesh can be extracted by

```
mesh.p(:, mesh.t(:, k))
```

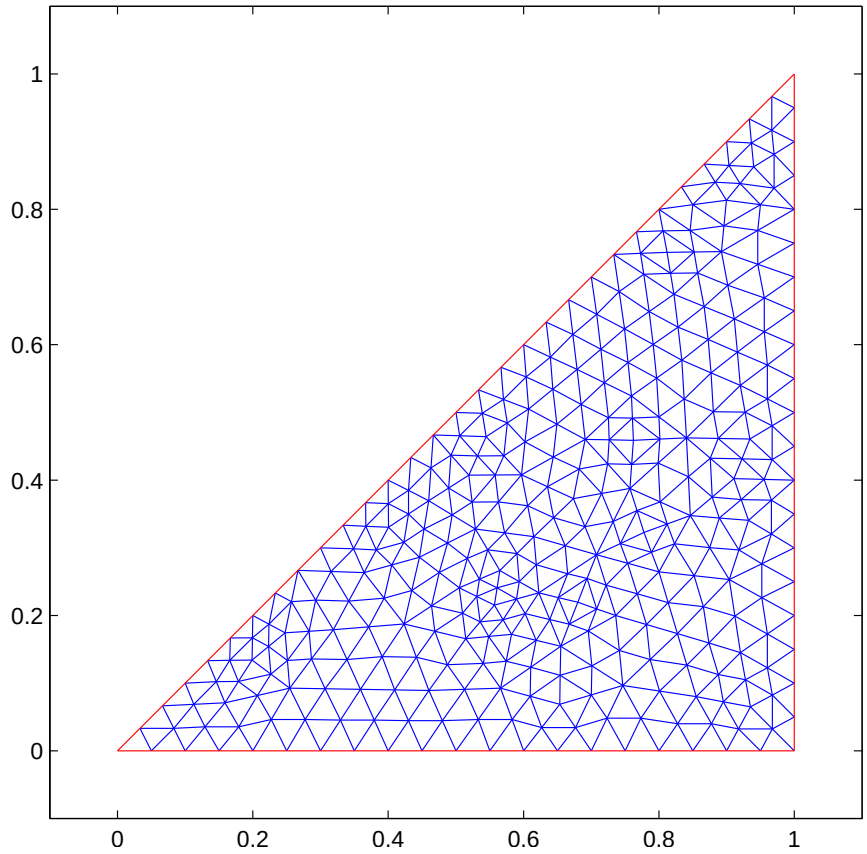
in which the first row contains the x coordinates for the three vertices and the second row contains the y coordinates.

Example: Plot a red $*$ at the center of the 1st element and the 2nd element of the previously generated mesh by different values for k in the script below:

```
k = 1; elem = mesh.p(:, mesh.t(:, k));
figure (3); hold on % assuming the mesh has be made in fig 3
center = (elem(:, 1) + elem(:, 2) + elem(:, 3))/3;
plot(center(1), center(2), 'r*')
```

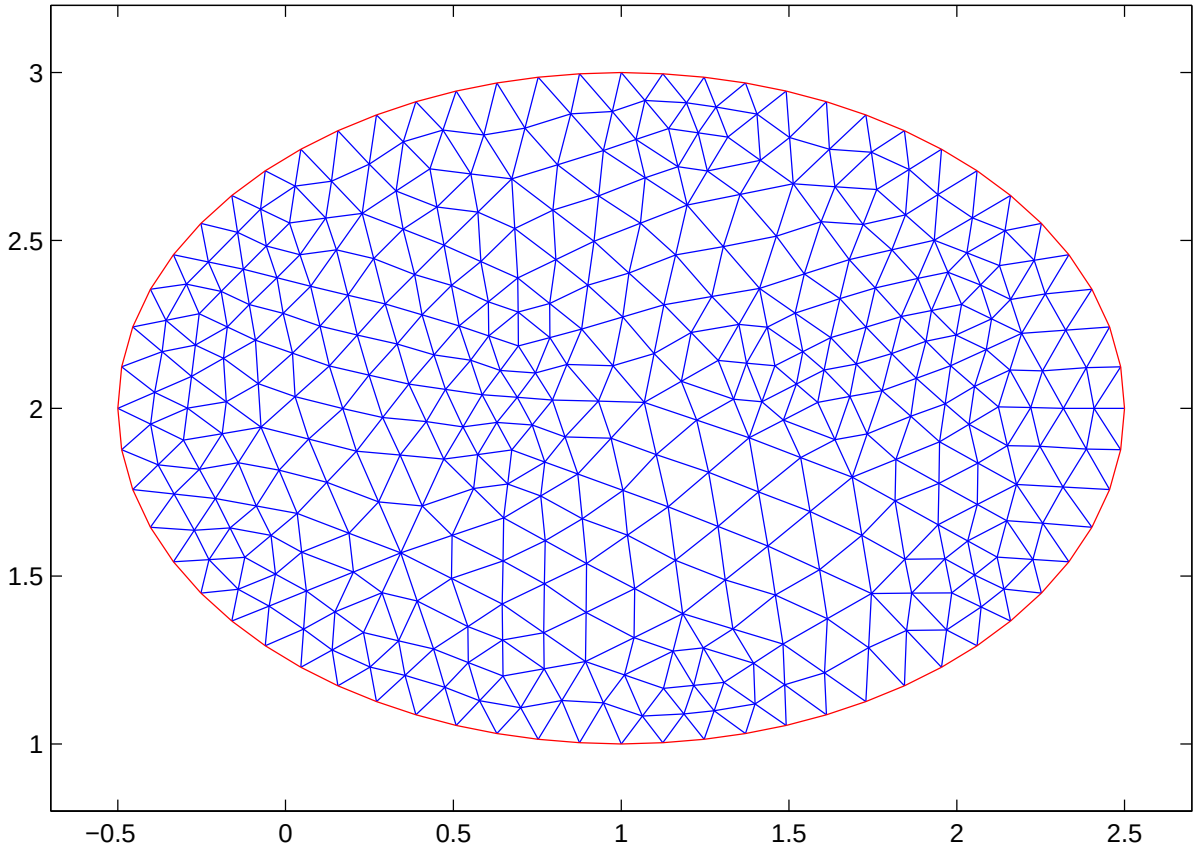
Here is a script to generate a mesh on a triangle/polygon.

```
C1 = [2, 3, 0, 1, 1, 0, 0, 1]'; gd = decsg(C1);
[p, e, t] = initmesh(gd); [p1, e1, t1] = refinemesh(gd, p, e, t);
mesh.p = p1; mesh.t = t1(1:3, :); figure(4); clf;
triplot(mesh.t', mesh.p(1, :)', mesh.p(2, :)');
axis([-0.1, 1.1, -0.1, 1.1]); axis equal
```



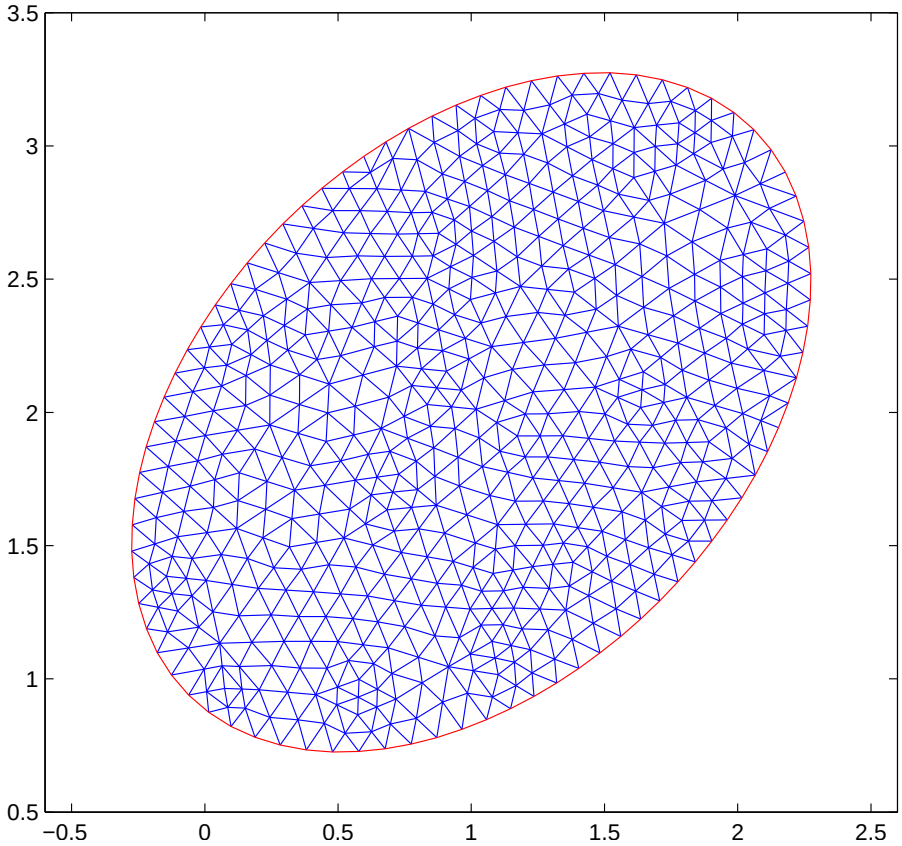
Here is a script to generate a mesh on an ellipse.

```
C1 = [4, 1, 2, 1.5, 1, 0]';
gd = decsg(C1);
[p, e, t] = initmesh(gd); [p1, e1, t1] = refinemesh(gd, p, e, t);
figure(5); clf;
pdemesh(p1, e1, t1); axis([-0.5, 2.5, 0.5, 3.5]); axis equal;
```



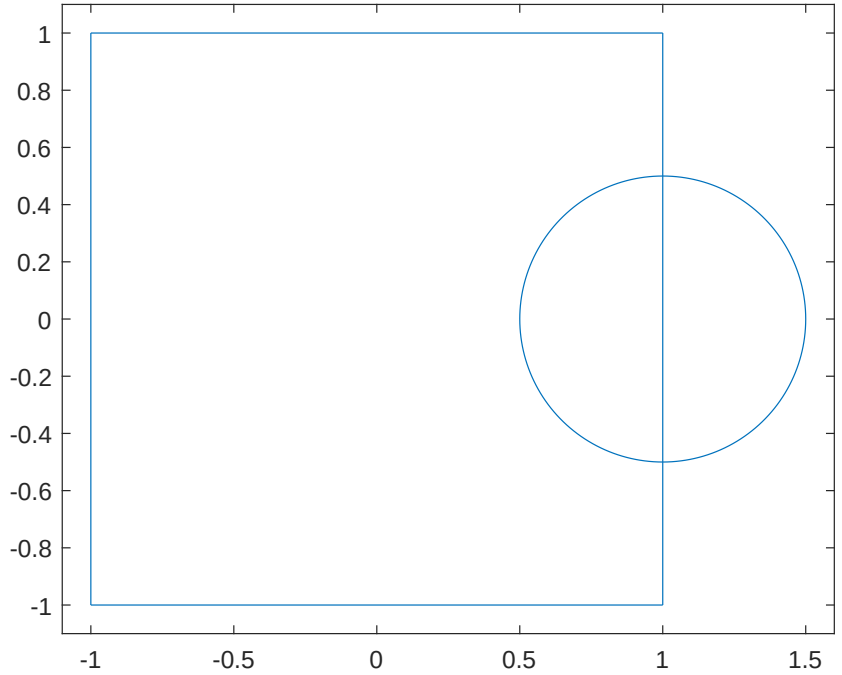
Here is a script to generate a mesh on a rotated ellipse.

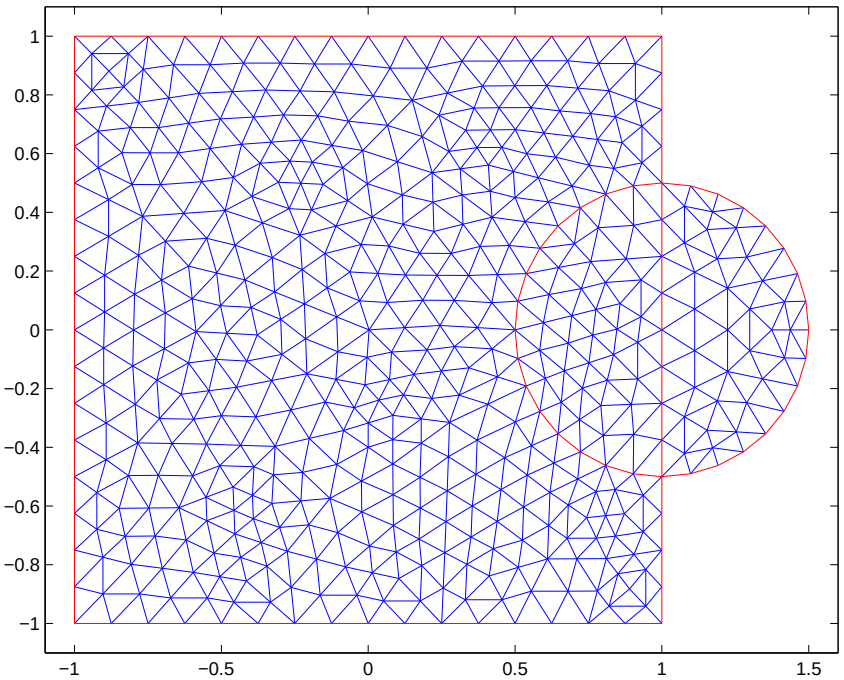
```
C1 = [4, 1, 2, 1.5, 1, pi/4]';
gd = decsg(C1);
[p, e, t] = initmesh(gd); [p1, e1, t1] = refinemesh(gd, p, e, t);
figure(6); clf;
pdemesh(p1, e1, t1); axis([-0.5, 2.5, 0.5, 3.5]); axis equal
```



A mesh for a domain formed as the union of a rectangle and a disk:

```
R1 = [3,4,-1,1,1,-1,-1,-1,1,1]'; C1 = [1,1,0,.5]';
% make C1 to have the same length as R1
C1 = [C1;zeros(length(R1)-length(C1),1)];
shapes = [R1, C1]; % combine the shapes into one matrix
ns = char('R1', 'C1'); ns = ns'; % name the basic shapes
sf = 'R1 + C1'; % set formula for the domain
gd = decsg(shapes, sf, ns); % geom data for the domain
figure(7); pdegplot(gd); axis([-1.1, 1.6, -1.1, 1.1]); % display the geom
[p, e, t] = initmesh(gd); [p, e, t] = refinemesh(gd, p, e, t);
figure(8); pdemesh(p, e, t); axis equal; axis([-1.1, 1.6, -1.1, 1.1]);
```



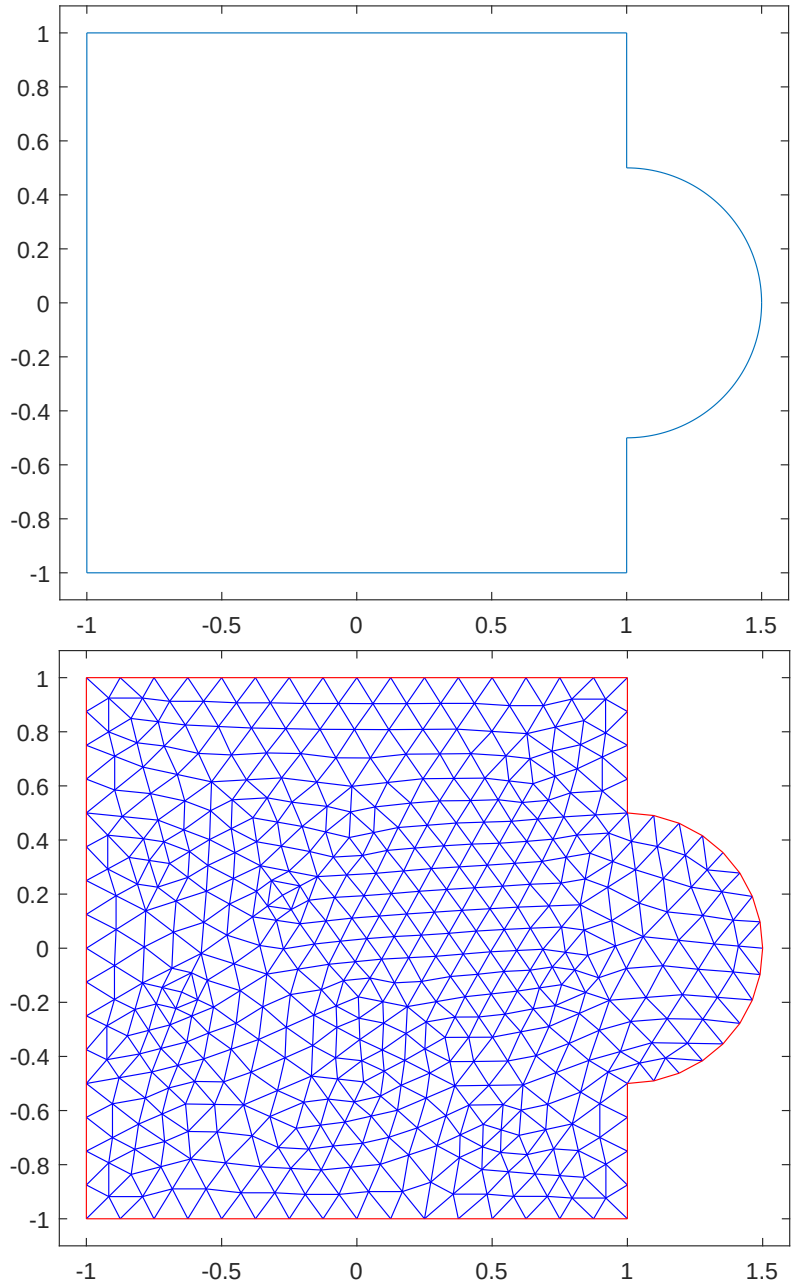


The domain and the mesh with the original boundaries

PDEtool's function `csgdel` can be used to delete the internal boundaries between subdomains for the solution domain.

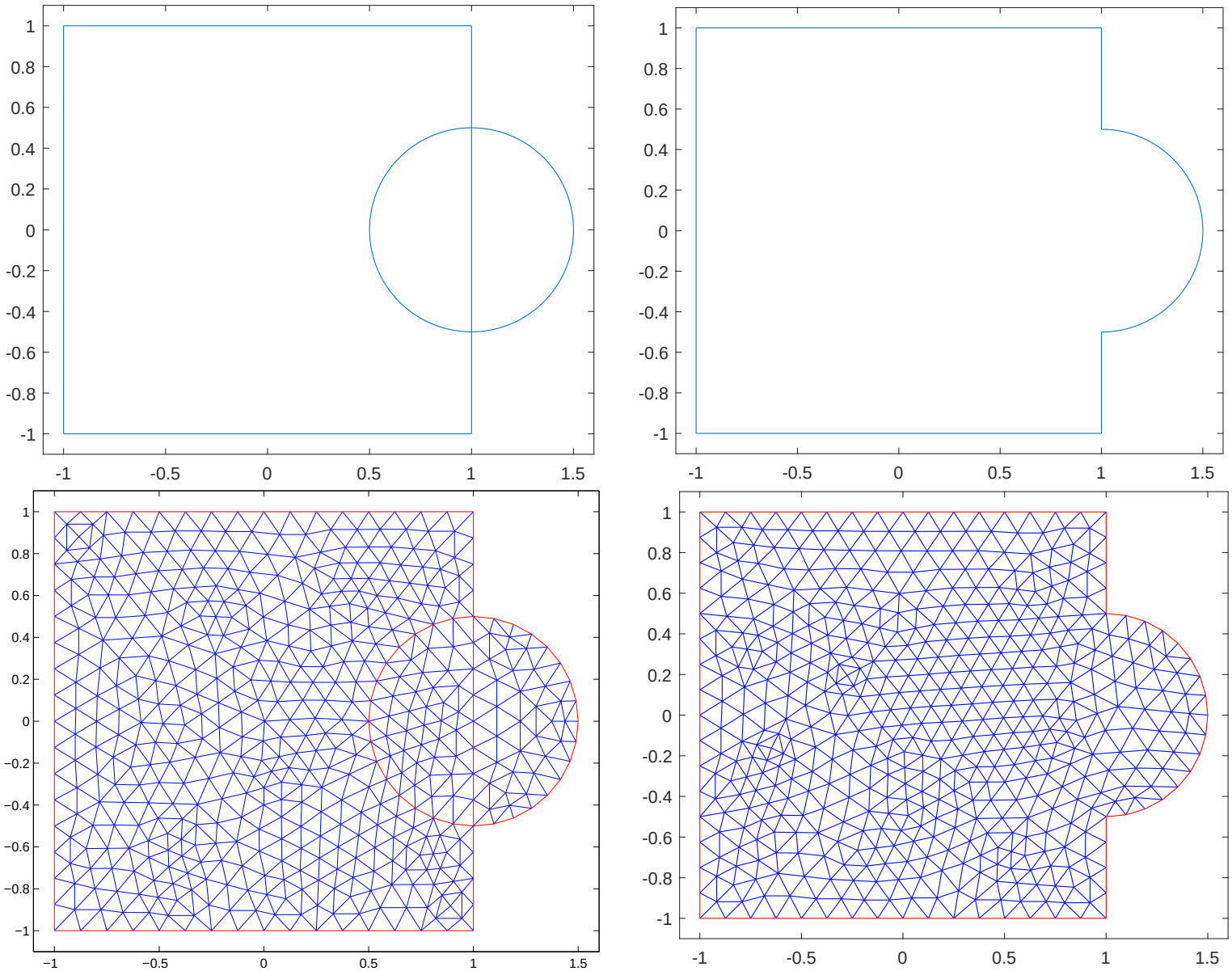
A mesh for the same domain as the previous one without internal boundaries/faces:

```
R1 = [3,4,-1,1,1,-1,-1,-1,1,1]'; C1 = [1,1,0,.5]';
% make C1 to have the same length as R1
C1 = [C1;zeros(length(R1)-length(C1),1)];
shapes = [R1, C1]; % combine the shapes into one matrix
ns = char('R1', 'C1'); ns = ns'; % name the basic shapes
sf = 'R1 + C1'; % set formula for the domain
[gd, bt] = decsg(shapes, sf, ns); % geometry data for the domain
[gd, bt] = csgdel(gd,bt); % delete the boundary inside the domain
figure(9); pdegplot(gd); axis([-1.1, 1.6, -1.1, 1.1]); % display the geom
[p, e, t] = initmesh(gd); [p1, e1, t1] = refinemesh(gd, p, e, t);
figure(10); pdemesh(p1, e1, t1); axis equal; axis([-1.1, 1.6, -1.1, 1.1]);
```



The domain and the mesh without the original boundaries

Compare two meshes in the same domain:



Here is an example to construct a mesh in a domain by decsg with a little more complicated geometry.

```
rect1 = [3, 4, ...
        -1, 1, 1, -1, ...
         0, 0, -0.5, -0.5]';
C1 = [1, 1, -0.25, 0.25]'; C2 = [1, -1, -0.25, 0.25]';
% Append extra zeros to the circles
% so they have the same number of rows as the rectangle
C1 = [C1;zeros(length(rect1) - length(C1),1)];
C2 = [C2;zeros(length(rect1) - length(C2),1)];
% put all the shapes into one matrix
shapes = [rect1,C1,C2];
% name all the shapes
s_names = char('rect1','C1','C2'); s_names = s_names';
sf = '(rect1 + C1) - C2'; % set formula for the domain
[domain, bt] = decsg(shapes, sf, s_names); % form the domain
figure(1); pdegplot(domain, 'EdgeLabels','on','FaceLabels','on');
axis([-1.2, 1.35, -0.6, 0.2])
[p, e, t] = initmesh(domain);
[p1, e1, t1] = refinemesh(domain, p, e, t);
figure(2); pdemesh(p1, e1, t1);
axis equal; axis([-1.2, 1.35, -0.6, 0.2])
```

We can use PDEtool's decsg to develop simpler geometry/domain generator. Here is an example for generating a rectangular domain.

```
function r = geom_rect(xmin, xmax, ymin, ymax)
R = [3, 4, xmin, xmax, xmax, xmin, ymin, ymin, ymax, ymax]';
r = decsg(R);
```

Similar Matlab functions can be packaged for other basic 2D solids.

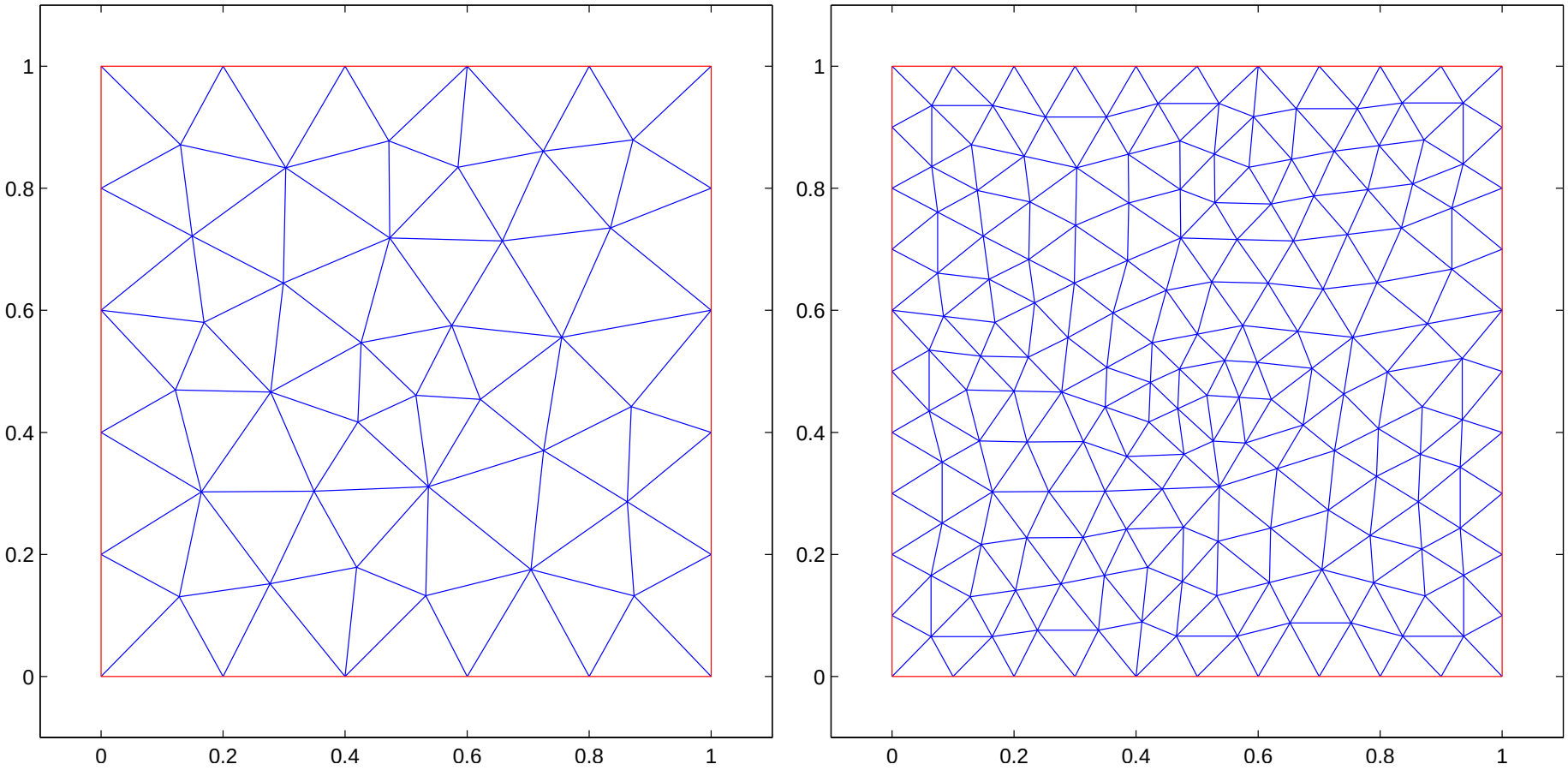
Here is a Matlab script for generating and visualizing a mesh for the unit square:

```
domain = geom_rect(0, 1, 0, 1);
figure(1); clf; pdegplot(domain)
axis([-0.1, 1.1, -0.1, 1.1])
[p, e, t] = initmesh(domain, 'Hmax', 0.2);
figure(2); clf; pdemesh(p, e, t)
axis([-0.1, 1.1, -0.1, 1.1])
```

This mesh can be refined once or more as follows:

```
[p1, e1, t1] = refinemesh(domain, p, e, t);
figure(3); clf; pdemesh(p1, e1, t1);
```

Remark: A popular uniform refinement technique for triangular meshes is to cut each triangular element into 4 congruent smaller triangles by connecting middle points on the edges.



Plotting a function on polygonal domain:

Plotting a function over a rectangular domain in Matlab is easy, but plotting a function over an arbitrary 2D domain in Matalab is not trivial. However, once we have a mesh for a 2D domain, then we can use Matlab's `trisurf` to plot a function on that domain

Plot a function on a triangular domain:

```
C1 = [2, 3, 0, 1, 1, 0, 0, 1]'; gd = decsg(C1);
[p, e, t] = initmesh(gd);
[p1, e1, t1] = refinemesh(gd, p, e, t);
mesh.p = p1; mesh.t = t1(1:3, :);
figure(1); clf; % display the mesh
triplot(mesh.t', mesh.p(1, :)', mesh.p(2, :)'); % show the mesh
axis([-0.1, 1.1, -0.1, 1.1]); axis equal

x = mesh.p(1, :); y = mesh.p(2, :);
phi = @(x, y) sin(2*pi*x).*(5*y); % the function to be plotted
z = phi(x, y); % evaluating the function on the mesh
figure(2); clf;
trisurf(mesh.t', x, y, z); % plot function on the mesh
figure(3); clf; trisurf(mesh.t', x, y, z); shading interp;
```

Plot a function on elliptical domain:

```
% domain and its mesh
C1 = [4, 1, 2, 1.5, 1, 0]';
gd = decsg(C1);
[p, e, t] = initmesh(gd);
[p1, e1, t1] = refinemesh(gd, p, e, t);
mesh.p = p1; mesh.t = t1(1:3, :);

% plot the mesh
figure(1); clf;
triplot(mesh.t', mesh.p(1, :)', mesh.p(2, :)'); axis equal;

% plot the function surface with the mesh
x = mesh.p(1, :); y = mesh.p(2, :);
phi = @(x, y) (x-1).^2/(1.5^2) + (y - 2).^2/(1^2) - 1;
z = phi(x, y);
figure(2); clf; trisurf(mesh.t', x, y, z);
figure(3); clf; trisurf(mesh.t', x, y, z); shading interp;
```

We can also use `pdeplot` from PDEtool box to plot a function. Here is is an example for how to use it:

```
pdeplot(p1, e1, t1, 'XYData', z, 'ZData', z, 'ColorBar', 'off')
axis equal
```

A mesh viewer: In developing finite element codes, it is sometimes helpful if we plot the mesh together with indices of elements and nodes. Here is a sample Matlab function for this purpose:

```
function FE_mesh_viewer_2D(mesh, nodes, elem_label, node_label)
% Usage: FE_mesh_viewer_2D(mesh, nodes, elem_label, node_label)
%
% elem_label = 0, no element index displayed,
% elem_label = 1, element index displayed,
%
% node_label = 0, no node index displayed
% node_label = 1, node index displayed
```

This function consists of two parts.

(1) First, use Matlab's `triplot` command to plot the mesh. If requested, the index of each element can also be plotted at the center of gravity for this element by the `text` command of Matlab.

(2) A do-loop for the `nodes` in which the code plots a $*$ at each node whose x- and y-coordinates are provided by `nodes`. If requested, the index of each node can also be plotted at/beside the node by the `text` command of Matlab.

Remark: The `nodes` inputs in this program is not necessarily the mesh nodes.

Some specifics for implementing the Matlab function:

```
function FE_mesh_viewer_2D(mesh, nodes, elem_label, node_label)
```

We will use the mesh data generated as follows to explain the implementation:

```
geom = geom_rect(0, 1, 0, 1);
[p, e, t] = initmesh(geom, 'Hmax', 0.9);
mesh = struct('p', p, 't', t(1:3, :));
```

Part (1): First, use Matlab's `triplot` command to plot the mesh:

```
triplot(mesh.t', mesh.p(1, :)', mesh.p(2, :));
```

Part (2): If `elem_label==1`, we loop over all the elements. For the k-th element, we can extract out its vertices, find the center, and display the index of the element at the center by

```
hold on;
for k = 1:size(mesh.t, 2) % loop over all elements
    vert = mesh.p(:, mesh.t(:, k)); % vertices of k-th element
    Cxy = (1/3)*(vert(:, 1) + vert(:, 2) + vert(:,3)); % center
    text(Cxy(1), Cxy(2), int2str(k)); % display index at the center
end
```

Part (3): If `node_label==1`, we loop over all the nodes. For the i-th node, we use its coordinates to display a star and the node index by

```
hold on
for i = 1:size(mesh.p, 2) % loop over all the nodes
    node = mesh.p(:, i); % i-th node
    plot(node(1), node(2), '*')
    text(node(1), node(2), int2str(i))
end
```

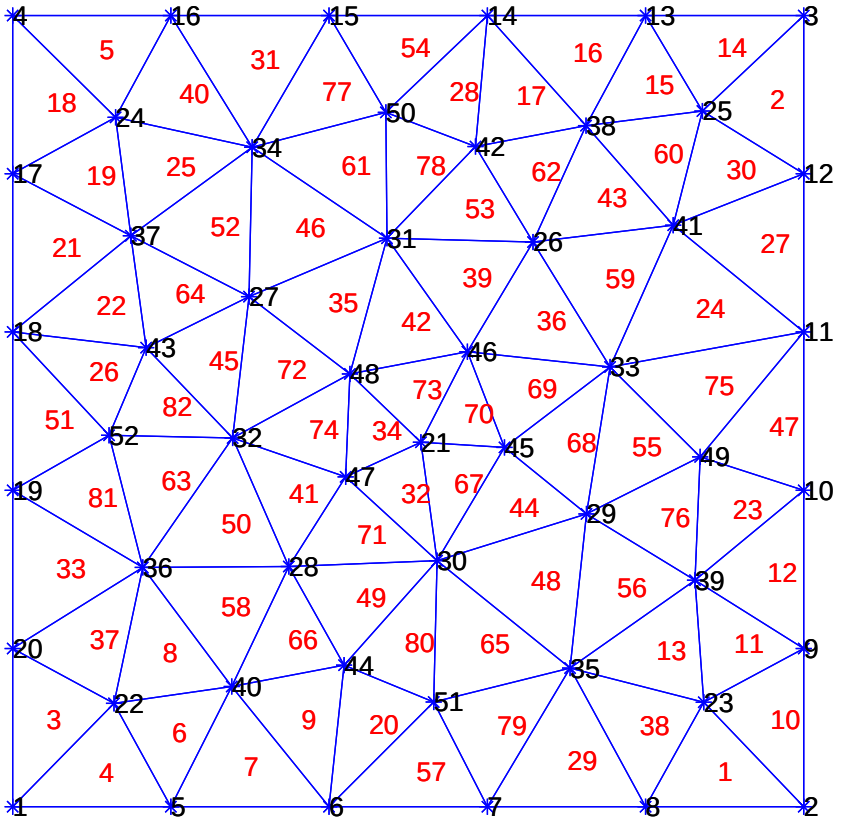
Remarks:

- The Matlab function `FE_mesh_viewer_2D` is a useful utility. Developing this function lets us to lean how to use parts in a `mesh` data structure.
- The Matlab function `FE_mesh_viewer_2D` to display edges indices for mesh structure that contains proper edge structure.

Here is a Matlab script to use this mesh plotter:

```
geom = geom_rect(0, 1, 0, 1);
[p, e, t] = initmesh(geom, 'Hmax', 0.2);
mesh = struct('p', p, 't', t(1:3, :));
elem_label = 1; % display elem indices
node_label = 1; % display node indices
FE_mesh_viewer_2D(mesh, mesh.p, elem_label, node_label)
```

which produces:



Remark: We can use the script and the graph on the previous slide to developt/debug the Matlab function `FE_mesh_viewer_2D`.

We can also use PDEtool's pdemesh command to plot a mesh together labels for nodes and elements. Here is a script demonstrating this:

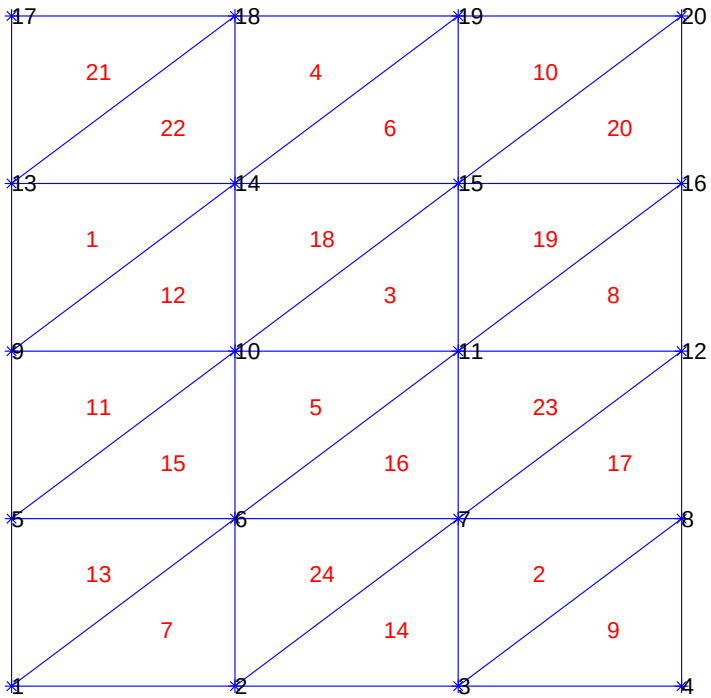
```
figure(2); clf;
geom = geom_rect(0, 1, 0, 1);
[p, e, t] = initmesh(geom, 'Hmax', 0.2);
pdemesh(p, e, t, 'NodeLabels', 'on', 'ElementLabels', 'on')
```

In summary, we have 3 ways to show a mesh:

- `FE_mesh_viewer_2D`
- `triplot`
- `pdemesh`

PDEtool's `poimesh` function can produce a Cartesian triangular mesh in a rectangle:

```
geom = geom_rect(0, 1, 0, 1);
[p, e, t] = poimesh(geom, 3, 4);
mesh = struct('p', p, 't', t(1:3, :));
figure(1); clf;
elem_label = 1; node_label = 1;
FE_mesh_viewer_2D(mesh, p, elem_label, node_label)
```



Remark: This script demonstrates that the input `nodes` does not have to be `mesh.p`.

Mesh-related data/functions in Matlab's PDEtool:

- Data formats for describing a domain with specific geometry such as circle, polygon, rectangle, and ellipse. Forming a composite domain with simple domains by set operations.
- Functions such as `decsg` and `csgdel` are for defining the geometry of a domain.
- Functions such as `initmesh` and `refinemesh` are for constructing a mesh with a given geometry, `poimesh` is for generating a regular mesh on a rectangular domain.
- Functions `pdegplot` and `pdemesh` are for visualizing the geometry of a domain and a mesh, respectively.
- `pdeplot` can be used to plot function with mesh.

Mesh-related data/functions in Matlab:

- `triplot` can be used to plot a mesh.
- `trisurf` can be used to plot a function defined on a mesh.

Remark: The mesh generators in PDEtool box has some randomness, especially about how the nodes and elements are ordered. For example, geometrically, the following script generate the same mesh, but p-t and $p1$-$t1$ should be considered as two different meshes such that the k-th element in both of them might be formed with different nodes.

```
geom = geom_rect(0, 1, 0, 1);

[p, e, t] = poimesh(geom, 3, 4);
mesh = struct('p', p, 't', t(1:3, :));
figure(1); clf;
pdemesh(p, e, t,  'NodeLabels', 'on', 'ElementLabels', 'on')
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;
figure(2); clf;
FE_mesh_viewer_2D(mesh, mesh.p, 1, 1)
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;

[p1, e1, t1] = poimesh(geom, 3, 4);
mesh1 = struct('p', p1, 't', t1(1:3, :));
figure(3); clf;
pdemesh(p1, e1, t1,  'NodeLabels', 'on', 'ElementLabels', 'on')
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;
figure(4); clf;
FE_mesh_viewer_2D(mesh1, mesh1.p, 1, 1)
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;
```

3.1.2, Generating Meshes Without PDE toolbox

Of course we can develop programs for generating meshes ourselves. Even though developing a mesh generator that can deal with domains with general geometries is a daunting software engineering project, we can develop programs for generating meshes of a domain with a simple geometry. Here is an example for producing a Cartesian triangular mesh in a rectangular domain.

A Cartesian triangular mesh generator for a rectangular domain:

```
function mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny)
% nx, ny: number of intervals in each direction
xnode = xmin:(xmax - xmin)/nx:xmax;
ynode = ymin:(ymax - ymin)/ny:ymax;
nnx = max(size(xnode));
nny = max(size(ynode));

% form nodes
p = zeros(2,nnx*nny); p_count = 1;
for j=1:nny
    for i=1:nnx
        p(:, p_count) = [xnode(i);ynode(j)];
        p_count = p_count + 1;
    end
end
```

```
% form elements
t = zeros(3, 2*nx*ny);
% form the first two element in the first rectangle
t(:,1) = [1; nnx+2; nnx+1]; t(:,2) = [1; 2; nnx+2]; t_count = 3;
for j=1:ny
    for i=2:nx % count rectangles in x direction
        t(:,t_count:t_count+1) = t(:,t_count - 2:t_count - 1) + 1;
        t_count = t_count + 2;
    end
    if j < ny % move the next row of rectangles
        t(:,t_count:t_count+1) = t(:,t_count - 2*nx:t_count - 2*nx+1) + nnx;
        t_count = t_count + 2;
    end
end
mesh = struct('p', p, 't', t);
```

Remark: This function can be modified for generating a rectangular mesh in a rectangle.
Example: Here is a script demonstrating how to use this mesh generator:

```
xmin = 0; xmax = 1; ymin = 0; ymax = 1;
nx = 3; ny = 5;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny)

figure(1); clf; elem_label = 1; node_label = 1;
FE_mesh_viewer_2D(mesh, mesh.p, elem_label, node_label)
```

3.1.3, Arrays for Edges in a Mesh

The `e` matrix from the output of the following mesh functions in PDEtool

```
[p, e, t] = initmesh(gd);
[p, e, t] = refinemesh(gd, p, e, t);
[p, e, t] = poimesh(geom, m, n);
```

do not provide information about all edges in the mesh. Fortunately, given a data structure `mesh` with `p` and `t` arrays, it is not hard to construct a useful data structures for edges in the mesh.

For example, we can generate three arrays for edges in a mesh:

- The `e` array: its k-th column lists the two nodes for the k-th edge.
- The `M_e` array: this is a sparse matrix, the meaning of its i-j entry will be explained later.
- The `M_ne` array: this is a sparse matrix, the meaning of its i-j entry will be explained later.

Assume `mesh` represents a triangular mesh. First, locally on the k-th element formed by three mesh nodes whose indices are given by `mesh.t(:, k)`, these 3 nodes/vertices are connected to each other in the k-th element, and this property can be described by the local connectivity matrix:

$$M_{e,k} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Globally, this mean the three nodes in the mesh

`mesh.t(1, k)`, `mesh.t(2, k)`, `mesh.t(3, k)`

are connected to each other to form three edges in the mesh.

We can introduce a matrix M_e of size $|\mathcal{N}_h| \times |\mathcal{N}_h|$ to describe the global connectivity of nodes in a mesh. Matrix M_e can be constructed by distributing $M_{e,k}$ to M_e for all $1 \leq k \leq |\mathcal{T}_h|$.

Example: Consider a mesh for $\Omega = [0,1] \times [0,1]$ generated by the following script:

```
xmin = 0; xmax = 1; ymin = 0; ymax = 1; nx = 2; ny = 2;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny);
figure(1); clf;
FE_mesh_viewer_2D(mesh, mesh.p, 1, 1)
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;
```

Recall that the mesh is such that

```
mesh.t =
     5     2     5     4     1     1     2     4
     9     3     6     8     5     2     6     5
     8     6     9     7     4     5     5     8
```

from which we know that this mesh has 9 nodes and 8 elements. We first let M_e be the 9×9 zeros matrix.

For $k = 1$, we can use `mesh.t(:, 1)` to modify M_e as follows:

$$M_e = \underset{\longrightarrow}{M_e\left(\begin{bmatrix}5\\9\\8\end{bmatrix}, \begin{bmatrix}5\\9\\8\end{bmatrix}\right) = M_e\left(\begin{bmatrix}5\\9\\8\end{bmatrix}, \begin{bmatrix}5\\9\\8\end{bmatrix}\right) + M_{e,1}} \quad M_e = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 2$, we can use `mesh.t(:, 2)` to modify M_e as follows:

$$M_e \left(\begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} \right) = M_e \left(\begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} \right) + M_{e,2} \longrightarrow M_e = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$M_e \left(\begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}, \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} \right) = M_e \left(\begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}, \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} \right) + M_{e,3} \longrightarrow$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 3$, we can use `mesh.t(:, 3)` to modify M_e as follows:

$$M_e \left(\begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}, \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} \right) = M_e \left(\begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}, \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} \right) + M_{e,3} \longrightarrow M_e = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 2 \end{bmatrix}$$

$$M_e \left(\begin{bmatrix} 4 \\ 8 \\ 7 \end{bmatrix}, \begin{bmatrix} 4 \\ 8 \\ 7 \end{bmatrix} \right) = M_e \left(\begin{bmatrix} 4 \\ 8 \\ 7 \end{bmatrix}, \begin{bmatrix} 4 \\ 8 \\ 7 \end{bmatrix} \right) + M_{e,4} \longrightarrow$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 4$, we can use `mesh.t(:, 4)` to modify M_e as follows:

$$M_e \quad M_e\left(\begin{bmatrix}4\\8\\7\end{bmatrix}, \begin{bmatrix}4\\8\\7\end{bmatrix}\right) = M_e\left(\begin{bmatrix}4\\8\\7\end{bmatrix}, \begin{bmatrix}4\\8\\7\end{bmatrix}\right) + M_{e,4} \quad \longrightarrow \quad M_e = \begin{bmatrix}0&0&0&0&0&0&0&0&0\\0&1&1&0&0&1&0&0&0\\0&1&1&0&0&1&0&0&0\\0&0&0&1&0&0&1&1&0\\0&0&0&0&2&1&0&1&2\\0&1&1&0&1&2&0&0&1\\0&0&0&1&0&0&1&1&0\\0&0&0&1&1&0&1&2&1\\0&0&0&0&2&1&0&1&2\end{bmatrix}$$

$$M_e\left(\begin{bmatrix}1\\5\\4\end{bmatrix}, \begin{bmatrix}1\\5\\4\end{bmatrix}\right) = M_e\left(\begin{bmatrix}1\\5\\4\end{bmatrix}, \begin{bmatrix}1\\5\\4\end{bmatrix}\right) + M_{e,5} \quad \longrightarrow$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 5$, we can use `mesh.t(:, 5)` to modify M_e as follows:

$$M_e \quad M_e\left(\begin{bmatrix}1\\5\\4\end{bmatrix},\begin{bmatrix}1\\5\\4\end{bmatrix}\right) = M_e\left(\begin{bmatrix}1\\5\\4\end{bmatrix},\begin{bmatrix}1\\5\\4\end{bmatrix}\right) + M_{e,5} \longrightarrow \quad M_e = \begin{bmatrix}1&0&0&1&1&0&0&0&0\\0&1&1&0&0&1&0&0&0\\0&1&1&0&0&1&0&0&0\\1&0&0&2&1&0&1&1&0\\1&0&0&1&3&1&0&1&2\\0&1&1&0&1&2&0&0&1\\0&0&0&1&0&0&1&1&0\\0&0&0&1&1&0&1&2&1\\0&0&0&0&2&1&0&1&2\end{bmatrix}$$

$$M_e\left(\begin{bmatrix}1\\2\\5\end{bmatrix},\begin{bmatrix}1\\2\\5\end{bmatrix}\right) = M_e\left(\begin{bmatrix}1\\2\\5\end{bmatrix},\begin{bmatrix}1\\2\\5\end{bmatrix}\right) + M_{e,6} \longrightarrow$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 6$, we can use `mesh.t(:, 6)` to modify M_e as follows:

$$M_e \quad M_e\left(\begin{bmatrix}1\\2\\5\end{bmatrix},\begin{bmatrix}1\\2\\5\end{bmatrix}\right) = M_e\left(\begin{bmatrix}1\\2\\5\end{bmatrix},\begin{bmatrix}1\\2\\5\end{bmatrix}\right) + M_{e,6} \longrightarrow \quad M_e = \begin{bmatrix}2&1&0&1&2&0&0&0&0\\1&2&1&0&1&1&0&0&0\\0&1&1&0&0&1&0&0&0\\1&0&0&2&1&0&1&1&0\\2&1&0&1&4&1&0&1&2\\0&1&1&0&1&2&0&0&1\\0&0&0&1&0&0&1&1&0\\0&0&0&1&1&0&1&2&1\\0&0&0&0&2&1&0&1&2\end{bmatrix}$$

$$M_e\left(\begin{bmatrix}2\\6\\5\end{bmatrix},\begin{bmatrix}2\\6\\5\end{bmatrix}\right) = M_e\left(\begin{bmatrix}2\\6\\5\end{bmatrix},\begin{bmatrix}2\\6\\5\end{bmatrix}\right) + M_{e,7} \longrightarrow$$

Recall `mesh.t` is the following matrix

```
5    2    5    4    1    1    2    4
9    3    6    8    5    2    6    5
8    6    9    7    4    5    5    8
```

For $k = 7$, we can use `mesh.t(:, 7)` to modify M_e as follows:

$$M_e \quad M_e\left(\begin{bmatrix}2\\6\\5\end{bmatrix}, \begin{bmatrix}2\\6\\5\end{bmatrix}\right) = M_e\left(\begin{bmatrix}2\\6\\5\end{bmatrix}, \begin{bmatrix}2\\6\\5\end{bmatrix}\right) + M_{e,7} \quad \longrightarrow \quad M_e = \begin{bmatrix}2&1&0&1&2&0&0&0&0\\1&3&1&0&2&2&0&0&0\\0&1&1&0&0&1&0&0&0\\1&0&0&2&1&0&1&1&0\\2&2&0&1&5&2&0&1&2\\0&2&1&0&2&3&0&0&1\\0&0&0&1&0&0&1&1&0\\0&0&0&1&1&0&1&2&1\\0&0&0&0&2&1&0&1&2\end{bmatrix}$$

$$M_e\left(\begin{bmatrix}4\\5\\8\end{bmatrix}, \begin{bmatrix}4\\5\\8\end{bmatrix}\right) = M_e\left(\begin{bmatrix}4\\5\\8\end{bmatrix}, \begin{bmatrix}4\\5\\8\end{bmatrix}\right) + M_{e,8} \quad \longrightarrow$$

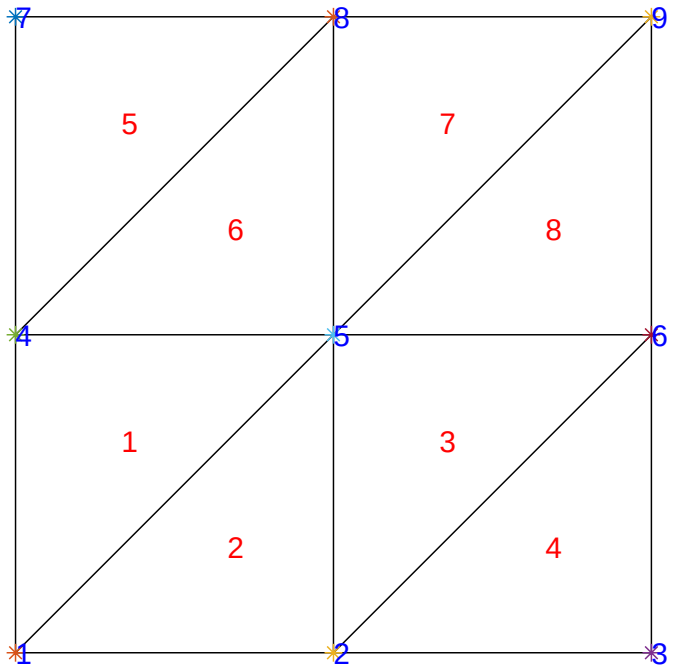
Recall `mesh.t` is the following matrix

5	2	5	4	1	1	2	4
9	3	6	8	5	2	6	5
8	6	9	7	4	5	5	8

For $k = 8$, we can use `mesh.t(:, 8)` to modify M_e as follows:

$$M_e \left(\begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix} \right) = M_e \left(\begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix} \right) + M_{e,8} \longrightarrow M_e$$

$$M_e = \begin{bmatrix} 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 & 2 & 0 & 1 & 2 & 0 \\ 2 & 2 & 0 & 2 & 6 & 2 & 0 & 2 & 2 \\ 0 & 2 & 1 & 0 & 2 & 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 2 \end{bmatrix}$$



In fact, matrix M_e tells us many things about the edges in a mesh among which we can easily form data structures useful for describing properties of edges in a mesh. Also, we note that matrix M_e is every sparse, especially for a fine mesh and we can use Matlab's `sparse` structure to represent it. Here is a sample Matlab function to generate the M_e matrix and other arrays:

```
function [e, M_e, M_ne] = edges_in_mesh(mesh, C, B)
n_edges = length(mesh.p) + length(mesh.t) - (2*C - B);
M_e = spalloc(length(mesh.p), length(mesh.p), n_edges);
for k = 1:length(mesh.t)
    M_e(mesh.t(:, k), mesh.t(:, k)) = ...
        M_e(mesh.t(:, k), mesh.t(:, k)) + ones(3, 3);
end
[I, J, S] = find(M_e);
% indices of the 2 nodes on an edge is from small to large
II = find(I<J);
e = [I(II)'; J(II)'];
M_ne = sparse(I(II), J(II), 1:length(II), length(mesh.p), ...
                length(mesh.p));
M_ne = M_ne + M_ne';
```

The inputs variable C is the number of connected components of Ω, B is the number of boundaries of Ω.

Remark: This program is inefficient because of the way how the sparse matrix `M_e` is constructed. More efficient construction of a sparse matrix will be discussed later.

Remark: Here, we have used the Euler-Poincaré formula for a triangular mesh

$$|\mathcal{E}_h| = |\mathcal{N}_h| + |\mathcal{T}_h| - (2C - B)$$

to estimate the number of edges, where C is the number of connected components of Ω, B is the number of boundaries of Ω.

Also, asymptotically, we have

$$|\mathcal{N}_h| : |\mathcal{T}_h| : |\mathcal{E}_h| \approx 1 : 2 : 3$$

.

How to use M_e:

- $M_e(i,i) = p,\ i = 1, 2, \cdots, |\mathcal{N}_h|$ means the i-th node is the vertex of p elements, i.e., there are p elements in the mesh which share the i-th node as a common vertex.
- When $i \neq j$, $M_e(i,j) \in \{0, 1, 2\}$, i.e., $0, 1, 2$ are the three possible values.
  - $M_e(i,j) = 0,\ i \neq j,\ i,j = 1, 2, \cdots, |\mathcal{N}_h|$ means the i-j nodes do not form an edge in the mesh.
  - $M_e(i,j) = p \neq 0,\ i \neq j,\ i,j = 1, 2, \cdots, |\mathcal{N}_h|$ means the i-j nodes form an edge which is shared by p elements. When $M_e(i,j) = p = 2$, the edge formed by the i-j nodes are share by 2 elements; hence, it must be an interior edge. When $M_e(i,j) = p = 1$, the edge formed by the i-j nodes is an edge of 1 element; hence, it must be an boundary edge.

How to use M_{ne}:

- $M_{ne}(i,j) = 0,\ i \neq j,\ i,j = 1, 2, \cdots, |\mathcal{N}_h|$ means the i-j nodes do not form an edge in the mesh.
- $M_{ne}(i,j) = p \neq 0,\ i \neq j,\ i,j = 1, 2, \cdots, |\mathcal{N}_h|$ means the i-j nodes form the p-th edge.

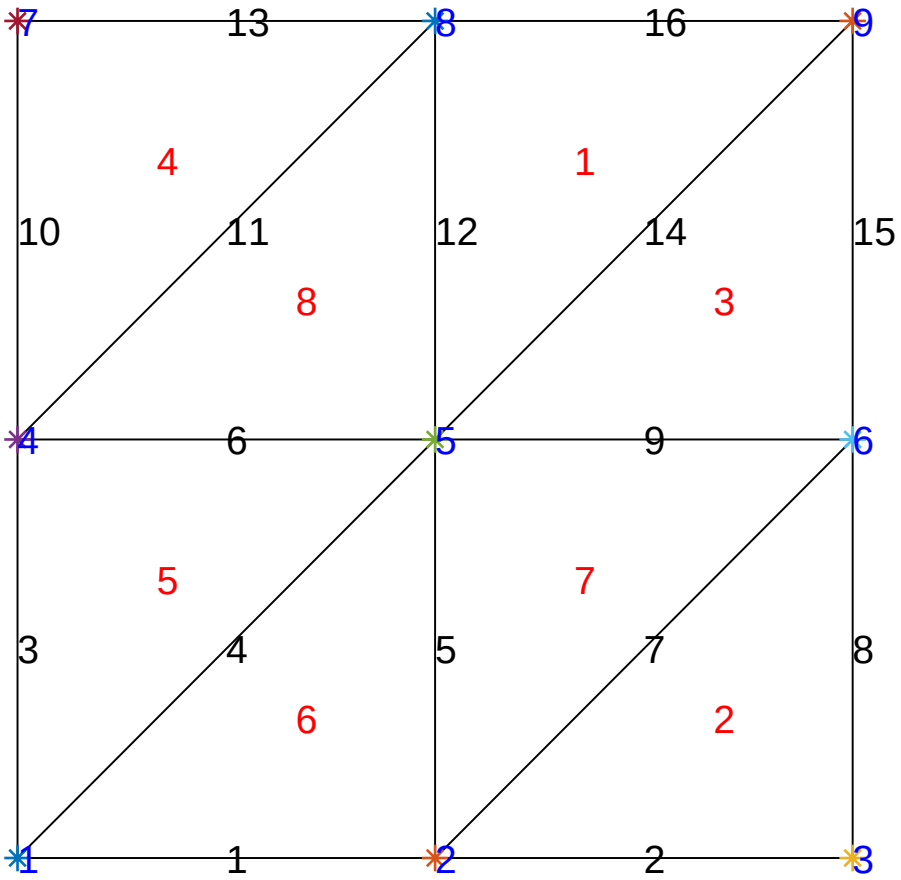
Here is a script to use the function edges_in_mesh.m:

```
geom = geom_rect(0, 1, 0, 1); % PDE toolbox
[p, e, t] = poimesh(geom, 2, 2); % PDE toolbox
mesh = struct('p', p, 't', t(1:3, :));
figure(1); clf;
FE_mesh_viewer_2D(mesh, mesh.p, 1, 1)
axis([-0.1, 1.1, -0.1, 1.1]); axis equal;

B = 1; % number of boundary
C = 1; % number of connected components
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
% extend the basic mesh
mesh.e = e; mesh.M_e = M_e; mesh.M_ne = M_ne;
```

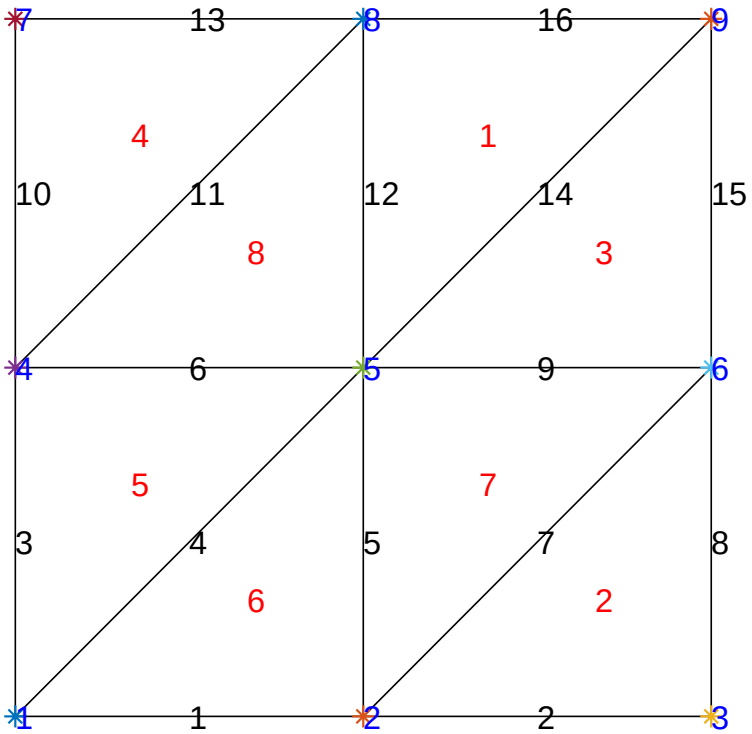
The edge array:

```
e =
1     2     1     1     2     4     2     3     5     4     4     5     7     5     6     8
2     3     4     5     5     5     6     6     6     7     8     8     8     9     9     9
```



For this mesh, M_e is the following array

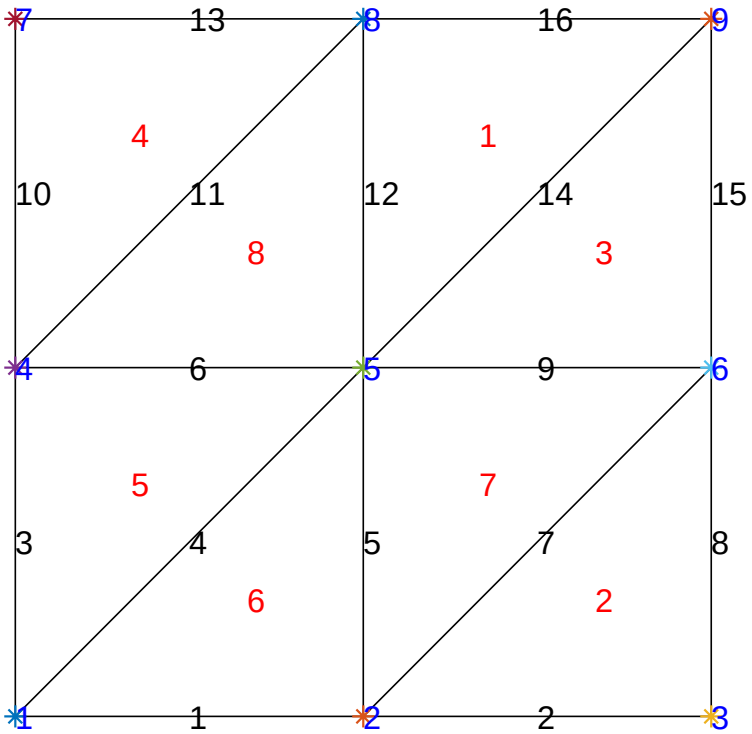
2	1	0	1	2	0	0	0	0
1	3	1	0	2	2	0	0	0
0	1	1	0	0	1	0	0	0
1	0	0	3	2	0	1	2	0
2	2	0	2	6	2	0	2	2
0	2	1	0	2	3	0	0	1
0	0	0	1	0	0	1	1	0
0	0	0	2	2	0	1	3	1
0	0	0	0	2	1	0	1	2



Since $M_e(3,3) = 1$, we know that the 3-rd node is the vertex of only 1 element. Since $M_e(5,5) = 6$, we know that the 5-th node is the vertex of 6 elements. Since $M_e(4,2) = 0$, nodes 2 and 4 do not form an edge in this mesh. Since $M_e(4,1) = 1$, we know that nodes 1 and 4 form one edge in the mesh and this edge belongs to only 1 element. This is a boundary edge. Since $M_e(4,8) = 2$, we know that nodes 4 and 8 form one edge in the mesh and this edge belongs to 2 elements. This is an interior edge.

For this mesh, M_{ne} is the following arrays

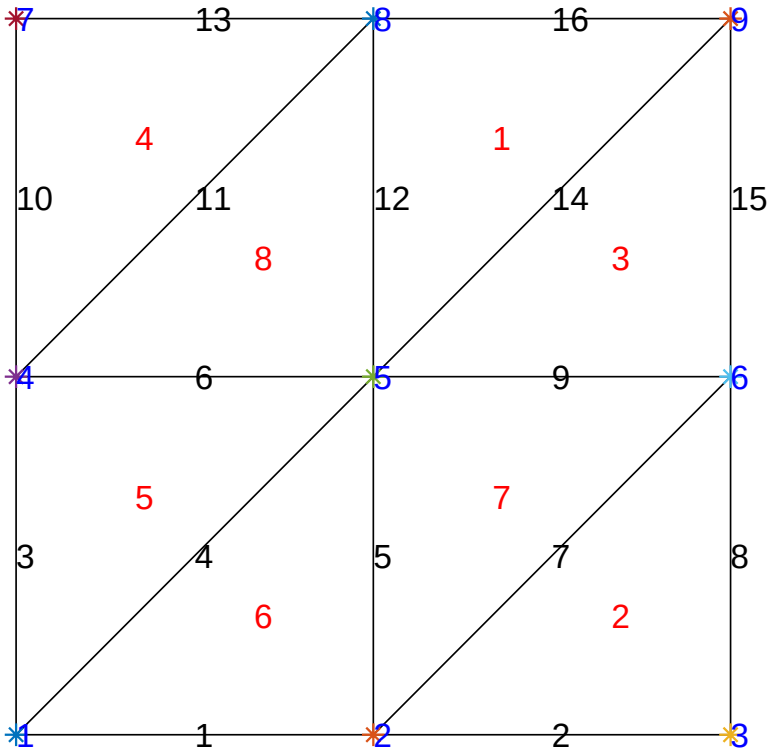
0	1	0	3	4	0	0	0	0
1	0	2	0	5	7	0	0	0
0	2	0	0	0	8	0	0	0
3	0	0	0	6	0	10	11	0
4	5	0	6	0	9	0	12	14
0	7	8	0	9	0	0	0	15
0	0	0	10	0	0	0	13	0
0	0	0	11	12	0	13	0	16
0	0	0	0	14	15	0	16	0



Since $M_{ne}(4,2) = 0$, nodes 2 and 4 do not form an edge in this mesh. Since $M_{ne}(4,1) = 3$, we know that nodes 1 and 4 form the 3-rd edge. Since $M_e(4,8) = 11$, we know that nodes 4 and 8 form the 11-th edge in the mesh. These conclusions can be confirmed by e array.

```
mesh.t =
     5     2     5     4     1     1     2     4
     9     3     6     8     5     2     6     5
     8     6     9     7     4     5     5     8
e =
1     2     1     1     2     4     2     3     5     4     4     5     7     5     6     8
2     3     4     5     5     5     6     6     6     7     8     8     8     9     9     9
```

$$M_{ne} = \begin{bmatrix} 0 & 1 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 5 & 7 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 6 & 0 & 10 & 11 & 0 \\ 4 & 5 & 0 & 6 & 0 & 9 & 0 & 12 & 14 \\ 0 & 7 & 8 & 0 & 9 & 0 & 0 & 0 & 15 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 11 & 12 & 0 & 13 & 0 & 16 \\ 0 & 0 & 0 & 0 & 14 & 15 & 0 & 16 & 0 \end{bmatrix}$$



Remark: The arrays `mesh.t, e, M_ne` facilitate computations on edges in an element-by-element way.

3.2, p-th Degree Lagrange Finite Element Spaces

Consider $\mathcal{T}_h$, either a triangular or a rectangular mesh, constructed on a domain Ω. We now discuss a Lagrange type finite element spaces defined on each element $K \in \mathcal{T}_h$. We will follow the affine equivalent finite elements idea by which these local finite element spaces are obtained by mapping corresponding finite element spaces constructed from a so called reference element $\hat{K}$ through an affine mapping.

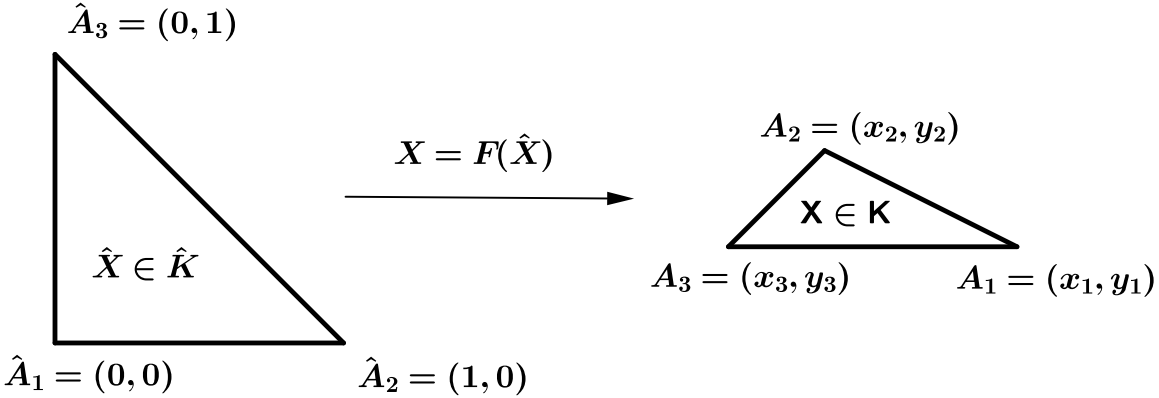
The reference triangular element $\hat{K}$: This is the triangle $\hat{K} = \triangle \hat{A}_1 \hat{A}_2 \hat{A}_3$ such that

$$\hat{A}_1 = (0,0)^t, \quad \hat{A}_2 = (1,0)^t, \quad \hat{A}_3 = (0,1)^t$$

The reference rectangular element $\hat{K}$: This is the rectangle $\hat{K} = \square \hat{A}_1 \hat{A}_2 \hat{A}_3 \hat{A}_4$ such that

$$\hat{A}_1 = (0,0)^t, \quad \hat{A}_2 = (1,0)^t, \quad \hat{A}_3 = (0,1)^t, \quad \hat{A}_4 = (1,1)^t$$

We note that vertices of the rectangular reference element are arranged in the lexicographical order which allow us to use the same affine transformation as that for triangular elements.



The affine mapping between $K \in \mathcal{T}_h$ and $\hat{K}$: Consider $K \in \mathcal{T}_h$ with

$$\begin{cases} K = \triangle A_1 A_2 A_3, & \text{when } \mathcal{T}_h \text{ is a triangular mesh} \\ K = \square A_1 A_2 A_3 A_4, & \text{when } \mathcal{T}_h \text{ is a rectangular mesh} \end{cases}$$

with $A_i = (x_i, y_i),\ 1 \le i \le 4$. Now, defined a mapping $F_K : \hat{K} \to \mathbf{R}^2$ such that

$$\begin{pmatrix} x \\ y \end{pmatrix} = F_K(\hat{x}, \hat{y}) = (A_2 - A_1, A_3 - A_1) \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + A_1 = B \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + A_1 \tag{3}$$

We can easily verify the following properties of this mapping.

- $F_K : \hat{K} \to \mathbf{R}^2$ is an affine mapping.
- $F_K(\hat{K}) = K$ with $F_K(\hat{A}_i) = A_i,\ 1 \le i \le 4$ in the sense that $(x, y) \in K$ with

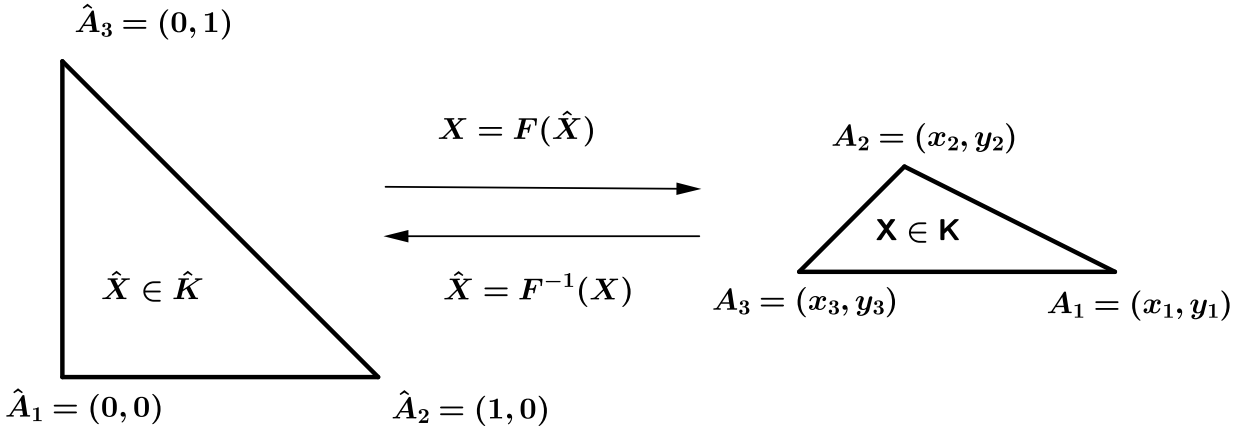
$$\begin{pmatrix} x \\ y \end{pmatrix} = F_K(\hat{x}, \hat{y}),\ \forall (\hat{x}, \hat{y}) \in \hat{K}$$

- The coefficient matrix $B = (A_2 - A_1, A_3 - A_1)$ of F_K is nonsingular provided that the element K is not degenerated, i.e., $|K| \neq 0$. Hence The inverse mapping $F_K^{-1} : K \to \hat{K}$ exists and

$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = F_K^{-1}(x, y) = B^{-1} \left[\begin{pmatrix} x \\ y \end{pmatrix} - A_1 \right] \tag{4}$$

- $F_K^{-1}(K) = \hat{K}$ with $F_K(A_i) = \hat{A}_i, 1 \leq i \leq 4$ in the sense that $(\hat{x}, \hat{y}) \in \hat{K}$ provided that

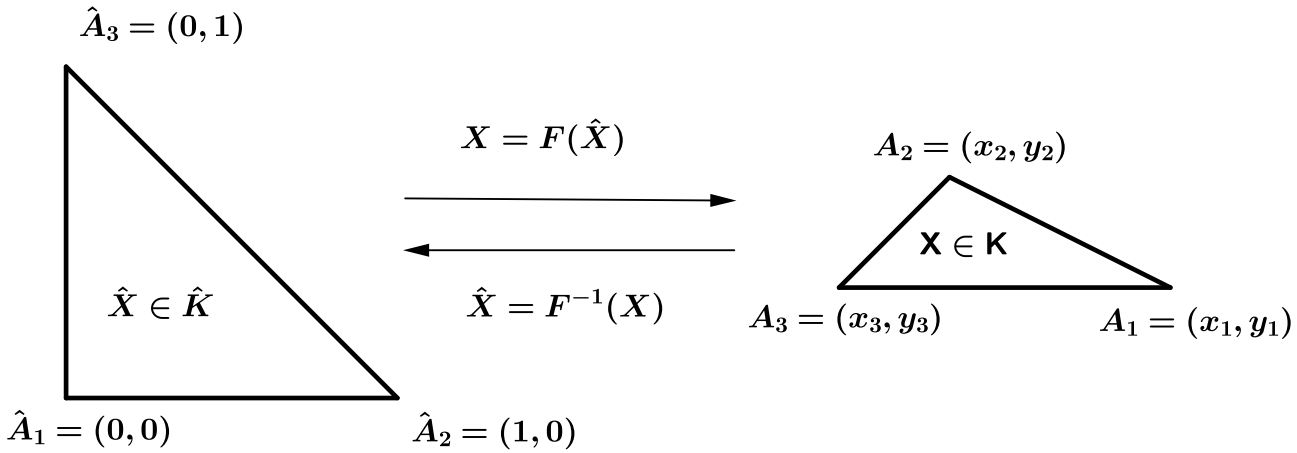
$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = F_K^{-1}(x, y), \ \forall (x, y) \in K$$



Some computational details of F_K and F_K^{-1}: For element $K = \triangle A_1A_2A_3$, recall that

$$B = (A_2 - A_1, A_3 - A_1) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}, \quad \hat{B} = B^{-1} = \begin{pmatrix} \hat{b}_{11} & \hat{b}_{12} \\ \hat{b}_{21} & \hat{b}_{22} \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = F_K(\hat{x}, \hat{y}) = B \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + A_1, \quad \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = F_K^{-1}(x, y) = B^{-1} \left[\begin{pmatrix} x \\ y \end{pmatrix} - A_1 \right] \tag{3,4}$$



Hence, $(x, y)^t = F_K(\hat{x}, \hat{y})$ and $(\hat{x}, \hat{y})^t = F_K^{-1}(x, y)$ can be computed as follows:

$$\begin{aligned} x &= x(\hat{x}, \hat{y}) = b_{11}\hat{x} + b_{12}\hat{y} + x_1 \\ y &= y(\hat{x}, \hat{y}) = b_{21}\hat{x} + b_{22}\hat{y} + y_1 \\ \hat{x} &= \hat{x}(x, y) = \hat{b}_{11}(x - x_1) + \hat{b}_{12}(y - y_1) \\ \hat{y} &= \hat{y}(x, y) = \hat{b}_{21}(x - x_1) + \hat{b}_{22}(y - y_1) \end{aligned}$$

For every (finite element) function $\hat{\phi}(\hat{x},\hat{y})$ constructed on $\hat{K}$, we can obtain a function $\phi(x,y)$ on K by letting

$$\begin{aligned}\phi(x,y) &= \hat{\phi}(F_K^{-1}(x,y)) = \hat{\phi}(\hat{x}(x,y),\hat{y}(x,y)) \\ &= \hat{\phi}(\hat{b}_{11}(x-x_1)+\hat{b}_{12}(y-y_1), \hat{b}_{21}(x-x_1)+\hat{b}_{22}(y-y_1)) \qquad (5)\end{aligned}$$

Recall that

$$\begin{aligned}\hat{x} = \hat{x}(x,y) &= \hat{b}_{11}(x-x_1)+\hat{b}_{12}(y-y_1) \\ \hat{y} = \hat{y}(x,y) &= \hat{b}_{21}(x-x_1)+\hat{b}_{22}(y-y_1)\end{aligned}$$

Derivatives of $\phi(x,y) = \hat{\phi}(\hat{x}(x,y),\hat{y}(x,y))$:

$$\begin{aligned}\phi_x(x,y) &= \hat{\phi}_{\hat{x}}(\hat{x}(x,y),\hat{y}(x,y))\hat{x}_x(x,y) + \hat{\phi}_{\hat{y}}(\hat{x}(x,y),\hat{y}(x,y))\hat{y}_x(x,y) \\ &= \hat{b}_{11}\hat{\phi}_{\hat{x}}(\hat{x}(x,y),\hat{y}(x,y)) + \hat{b}_{21}\hat{\phi}_{\hat{y}}(\hat{x}(x,y),\hat{y}(x,y)) \qquad (6) \\ \phi_y(x,y) &= \hat{\phi}_{\hat{x}}(\hat{x}(x,y),\hat{y}(x,y))\hat{x}_y(x,y) + \hat{\phi}_{\hat{y}}(\hat{x}(x,y),\hat{y}(x,y))\hat{y}_y(x,y) \\ &= \hat{b}_{12}\hat{\phi}_{\hat{x}}(\hat{x}(x,y),\hat{y}(x,y)) + \hat{b}_{22}\hat{\phi}_{\hat{y}}(\hat{x}(x,y),\hat{y}(x,y)) \qquad (7)\end{aligned}$$

Formulas for higher order derivatives of $\phi(x,y) = \hat{\phi}(\hat{x}(x,y),\hat{y}(x,y))$ can be derived similarly.

3.2.1, The p-th degree finite element space on a triangular element $K \in \mathcal{T}_h$: First, we consider the construction of the p-th degree finite element space $V^p(\hat{K})$ on the reference triangular element $\hat{K} = \triangle \hat{A}_1 \hat{A}_2 \hat{A}_3$. Simply,

$$V^p(\hat{K}) = \Pi_p$$

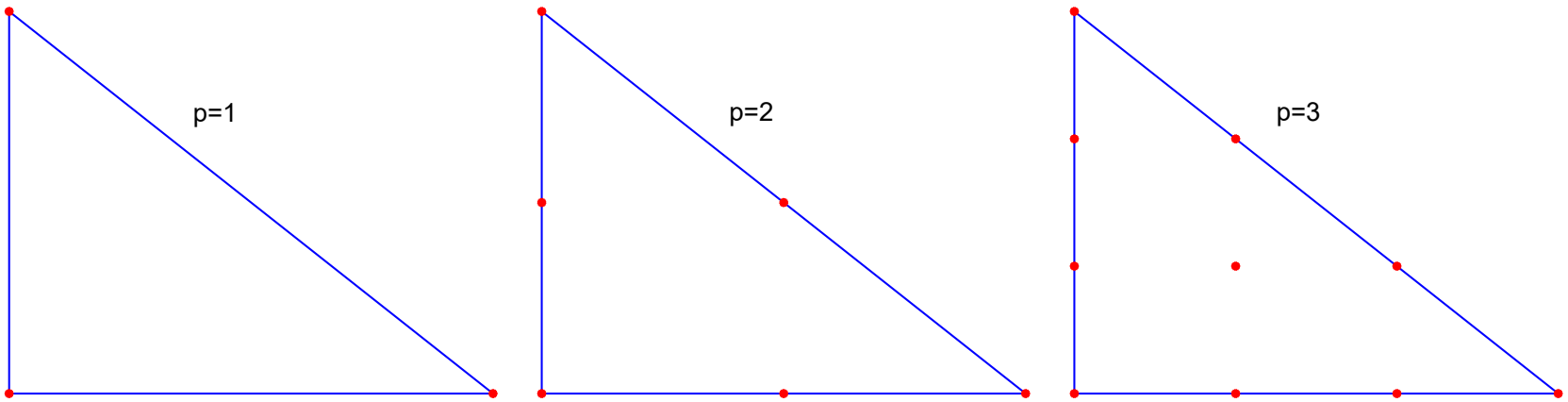
where Π_p is the set of two dimensional polynomials of degree up to p. For each integer $p \geq 0$, the polynomial space Π_p can be described by the Pascal's triangle as follows:

Π_0				1				
Π_1			x		y			
Π_2		x^2		xy		y^2		
Π_3	x^3		x^2y		xy^2		y^3	
Π_4	x^4	x^3y		x^2y^2		xy^3		y^4
$\vdots$								

- Π_p listed on each row in the table above is the linear space spanned by the monomials above that row.
- $dim(\Pi_p) = 1 + 2 + \cdots + (p+1) = \frac{(p+1)(p+2)}{2}$.

Shape functions of $V^p(\hat{K})$: Even though $V^p(\hat{K}) = \Pi_p$ on the reference elements, those monomials spanning Π_p are not convenient for finite element computations. Instead, we will use the shape functions to describe $V^p(\hat{K})$.

p-th degree Lagrange nodes on $\hat{K}$ for $V^p(\hat{K})$: Assume $p \geq 1$. On the reference triangle $\hat{K} = \hat{A}_1\hat{A}_2\hat{A}_3$, we introduce $p+1$ equally spaced parallel lines starting from $\overline{\hat{A}_1\hat{A}_2}$. These lines intersect with $\overline{\hat{A}_3\hat{A}_1}$ and $\overline{\hat{A}_3\hat{A}_2}$ to form $p+1$ line segments, labeled as $l_{p+1}, l_p, l_{p-1}, \cdots, l_2, l_1$. For $k = p+1, p, \cdots, 2, 1$, we introduce k equally spaces points including the end points on line l_k, and we call these points the Lagrange nodes on the reference element $\hat{K}$ for the finite element space $V^p(\hat{K})$.

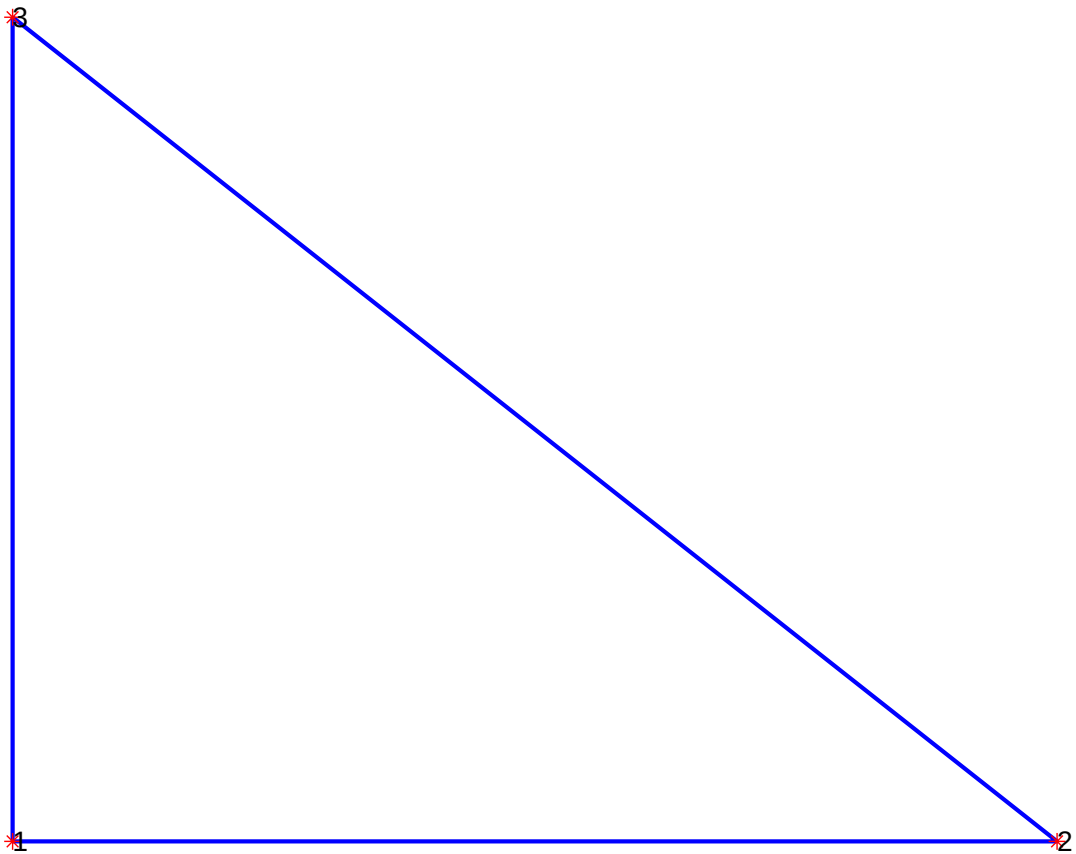


We order the Lagrange nodes on the reference element $\hat{K}$ as

$$\hat{A}_i = (\hat{x}_i, \hat{y}_i), \quad i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

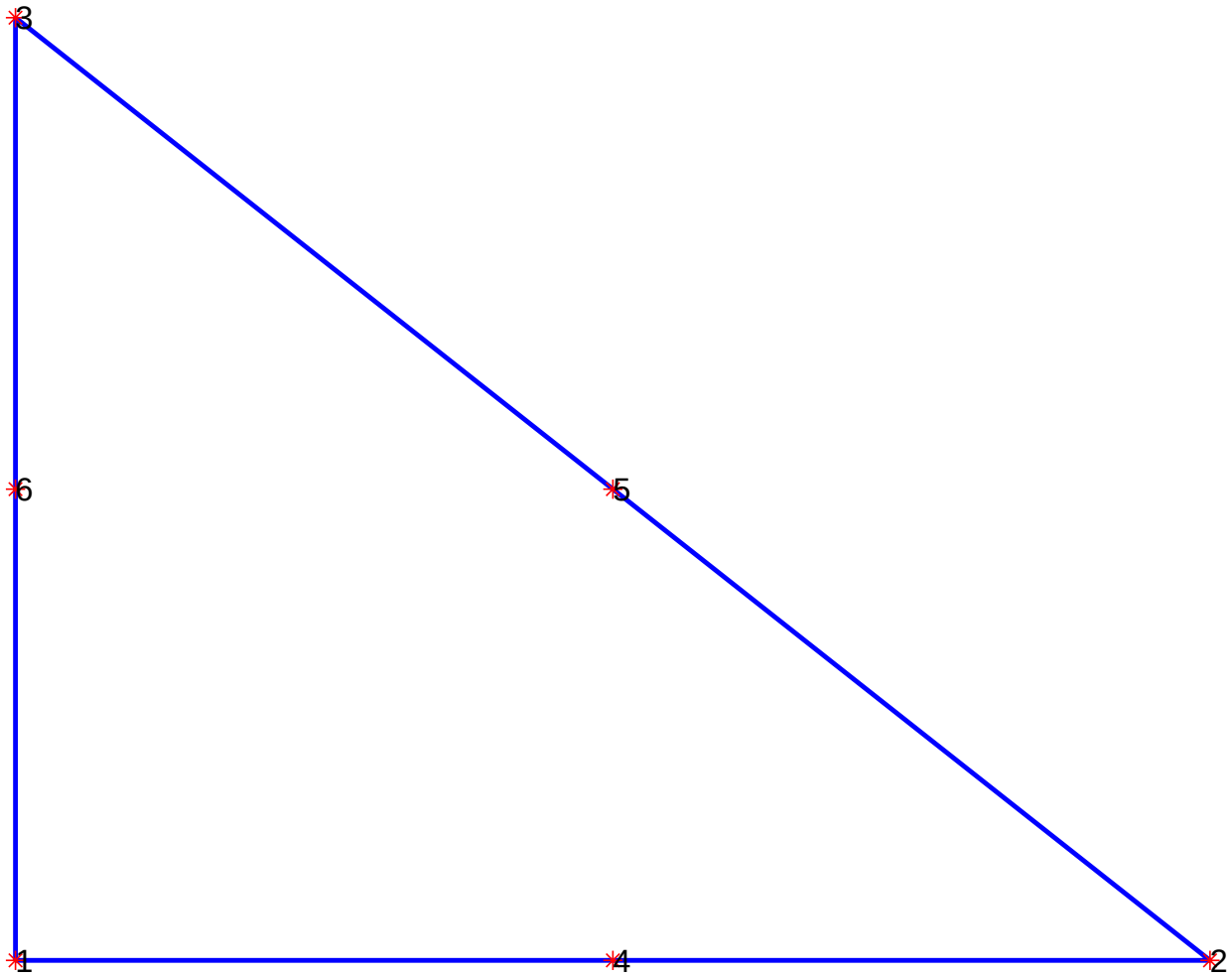
Again, how to order these nodes is not critical, but we need to order them and follow this order in all the computations later. The following are some conventional ways to order the Lagrange nodes for $V^p(\hat{K})$.

For $V^1(\hat{K})$: $\hat{A}_1 = (0, 0)$, $\hat{A}_2 = (1, 0)$, $\hat{A}_3 = (0, 1)$



For $V^2(\hat{K})$: $\hat{A}_1 = (0,0),\ \hat{A}_2 = (1,0),\ \hat{A}_3 = (0,1)$

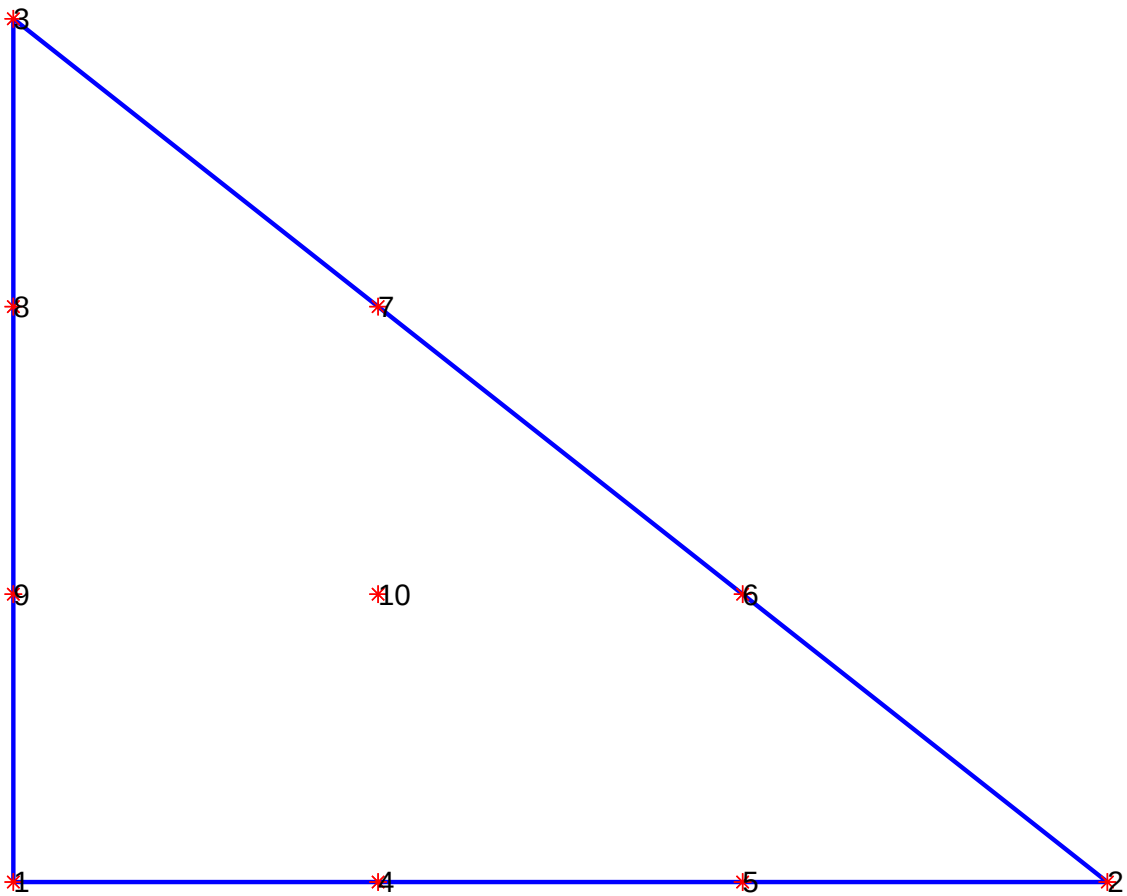
$$\hat{A}_4 = (1/2,0),\ \hat{A}_5 = (1/2,1/2),\ \hat{A}_6 = (0,1/2)$$



For $V^3(\hat{K})$: $\hat{A}_1 = (0,0),\ \hat{A}_2 = (1,0),\ \hat{A}_3 = (0,1)$

$$\hat{A}_4 = (1/3,0),\ \hat{A}_5 = (2/3,0),\ \hat{A}_6 = (2/3,1/3),\ \hat{A}_7 = (1/3,2/3)$$
$$\hat{A}_8 = (0,2/3),\ \hat{A}_9 = (0,1/3)$$

$$\hat{A}_{10} = (1/3,1/3)$$



Local degrees of freedom for $V_h^p(\hat{K})$: For any function $\hat{\phi} \in V^p(\hat{K})$, we call

$$\hat{\phi}(\hat{A}_i), \quad i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

its local degrees of freedom.

Recall: A hyperplane is a surface/line of a linear polynomial.

Lemma 1.1

Let p be a polynomial of degree $k \geq 1$ that vanishes on a hyperplane described by linear polynomial $L(X)$. Then $p(X) = L(X)Q(X)$ where $deg(Q) \leq k - 1$.

Proof: We give a proof for two dimension. The arguments can be easily extended to other dimensions. Without loss of generality, we assume that

$$L(X) = (E^{\perp})'(X - A)$$

where E is a unit vector and $E^{\perp}$ is the unit vector orthogonal to E. Then, using the affine transformation

$$X = A + (E, E^{\perp}) \begin{pmatrix} \xi \\ \eta \end{pmatrix}$$

we have

$$L(\xi, \eta) = L(X) = (E^{\perp})'(X - A) = (E^{\perp})'(E, E^{\perp}) \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \eta$$

This means that, in the ξ-η coordinates, the hyperplane is the ξ-axis: $\eta = 0$, and

$$p(\xi, \eta) = p(X) = \sum_{j=0}^{k} \sum_{i \le k-j} c_{ij} \xi^i \eta^j$$

Since $p(X)$ vanishes on the hyperplane, we have

$$0 \quad = \quad p(\xi, 0) = \sum_{i \le k} c_{i0} \xi^i \quad \forall \xi$$

from which we have $c_{i0} = 0, i = 0, 1, \cdots, k$. Hence

$$\begin{aligned} p(\xi, \eta) &= \sum_{j=0}^{k} \sum_{i \le k-j} c_{ij} \xi^i \eta^j \\ &= \sum_{j=1}^{k} \sum_{i \le k-j} c_{ij} \xi^i \eta^j = \eta \sum_{j=1}^{k} \sum_{i \le k-j} c_{ij} \xi^i \eta^{j-1} \\ &= \eta Q(\xi, \eta) = L(X) Q(X) \end{aligned}$$

with $deg(Q(X)) \le k - 1$.

Theorem 1.2

(Unisolvence for Shape Functions) For every integer $p \geq 1$ and a set of numbers $v_i, \quad i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$, there exists a unique polynomial $\hat{\phi} \in V^p(\hat{K})$ such that

$$\hat{\phi}(\hat{A}_i) = v_i, \ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

Proof: Note that $V^p(\hat{K}) = \Pi_p$. Hence $\hat{\phi} \in V^p(\hat{K})$ can be written as

$$\hat{\phi}(\hat{x}, \hat{y}) = \sum_{k=0}^{p} \sum_{j=0}^{p-k} c_{kj} \hat{x}^j \hat{y}^k$$

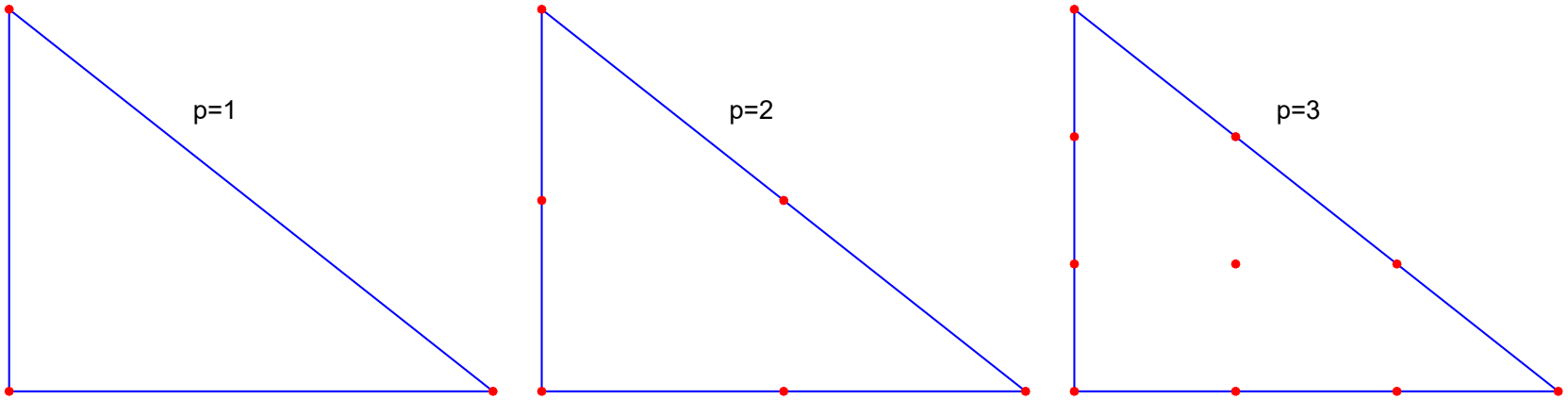
Then $\hat{\phi}(\hat{A}_i) = v_i, \ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$ are linear equations about the coefficients c_{kj}s of $\hat{\phi}$. Therefore we only need to prove the uniqueness. Hence, we assume

$$\hat{\phi}(\hat{A}_i) = 0, \ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

Then, we can use Lemma 1.1 to prove that $\hat{\phi}$ is a zero polynomial ???

Remark: In fact, from the proof for the previous theorem, we can see each p-degree polynomial is uniquely determined by its values at the p-th degree Lagrange nodes in the reference triangle $\hat{K}$.

Recall: p-th degree Lagrange nodes on the reference element $\hat{K} = \triangle \hat{A}_1 \hat{A}_2 \hat{A}_3$:



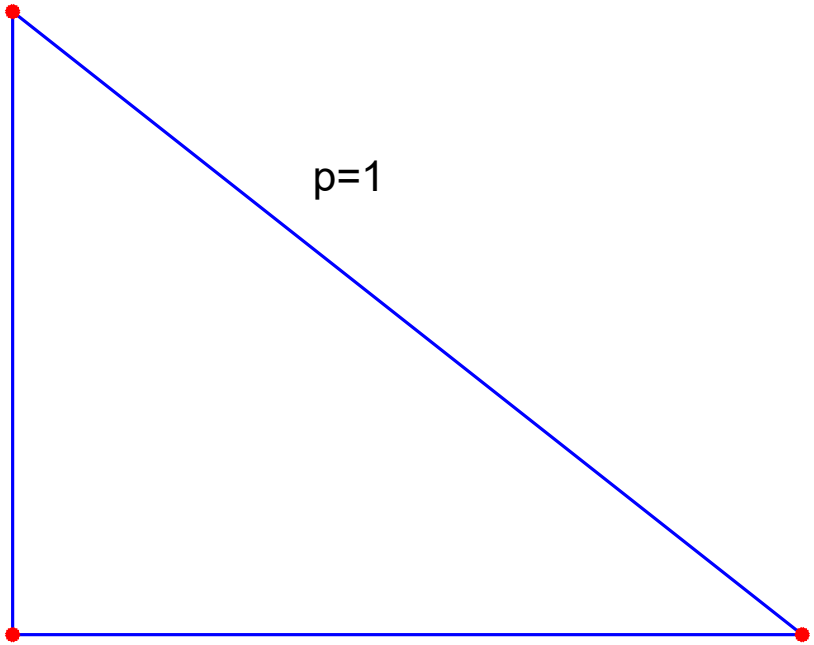
Theorem 1.2 implies that, for each $i = 1, 2, \cdots, (p+1)(p+2)/2$, there exists a unique p-th degree polynomial $\hat{\phi}_{i,p}(\hat{X}) \in V^p(\hat{K})$ such that

$$\hat{\phi}_{i,p}(\hat{A}_j) = \delta_{ij}, \quad j = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

The formulas for $\hat{\phi}_{i,p}(\hat{A}_j), \quad j = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$???

Formulas for shape functions on the reference element $\hat{K}$: We now discuss the shape functions for $V^p(\hat{K})$ for $p = 1, 2, 3$.

Formulas for 1st degree shape functions:



$$\begin{aligned}\hat{\phi}_{1,1}(\hat{x},\hat{y}) &= 1-\hat{x}-\hat{y}\\ \hat{\phi}_{2,1}(\hat{x},\hat{y}) &= \hat{x}\\ \hat{\phi}_{3,1}(\hat{x},\hat{y}) &= \hat{y}\end{aligned}$$

The first order partial derivatives of these shape function:

$$\hat{\phi}_{1,1,\hat{x}}(\hat{x},\hat{y}) = -1, \quad \hat{\phi}_{1,1,\hat{y}}(\hat{x},\hat{y}) = -1$$
$$\hat{\phi}_{2,1,\hat{x}}(\hat{x},\hat{y}) = 1, \quad \hat{\phi}_{2,1,\hat{y}}(\hat{x},\hat{y}) = 0$$
$$\hat{\phi}_{3,1,\hat{x}}(\hat{x},\hat{y}) = 0, \quad \hat{\phi}_{3,1,\hat{y}}(\hat{x},\hat{y}) = 1$$

These first degree shape functions are very important in the sense that they can be used to construct higher degree shape functions.

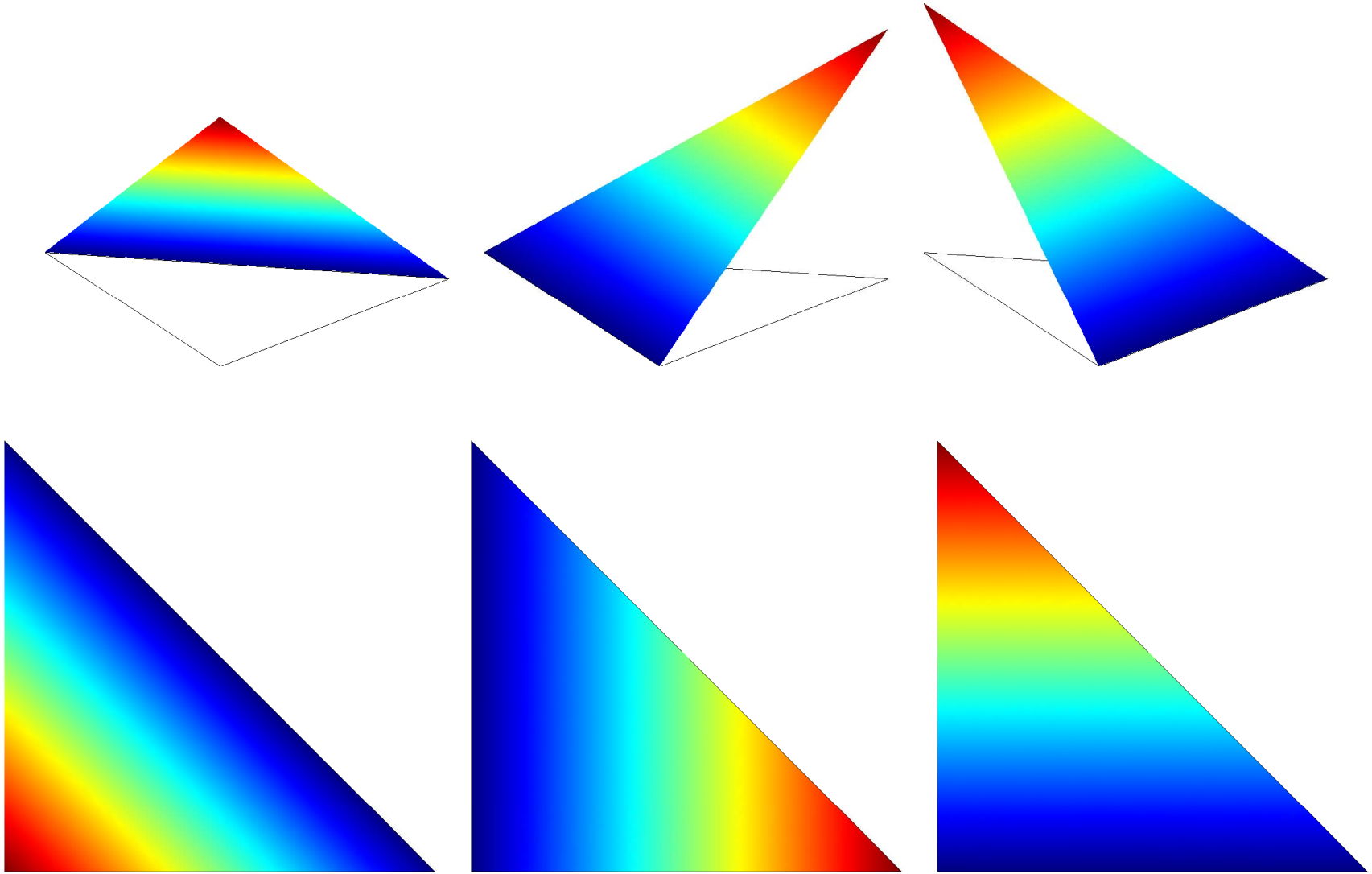
```
function phih = shape_fun_2D_Lagrange_tri_ref_degree1(hx, hy, shape_index, ...
                    d_hx, d_hy)
phih =  zeros(size(hx));
if d_hx == 0 && d_hy == 0
    if shape_index == 1
        phih =  1 - hx - hy;
    elseif shape_index == 2
        phih =  hx;
    elseif shape_index == 3
        phih =  hy;
    end
elseif d_hx == 1 && d_hy == 0
    if shape_index == 1
        phih =  -ones(size(hx));
    elseif shape_index == 2
        phih =  ones(size(hx));
    elseif shape_index == 3
        phih =  zeros(size(hx));
    end
elseif d_hx == 0 && d_hy == 1
    if shape_index == 1
        phih =  -ones(size(hx));
    elseif shape_index == 2
        phih =  zeros(size(hx));
    elseif shape_index == 3
        phih =  ones(size(hx));
    end
else
    phih =  zeros(size(hx));
end
```

Formulas for 1st degree shape functions:

$$\begin{aligned}\hat{\phi}_{1,1}(\hat{x},\hat{y}) &= 1-\hat{x}-\hat{y}\\ \hat{\phi}_{2,1}(\hat{x},\hat{y}) &= \hat{x}\\ \hat{\phi}_{3,1}(\hat{x},\hat{y}) &= \hat{y}\end{aligned}$$

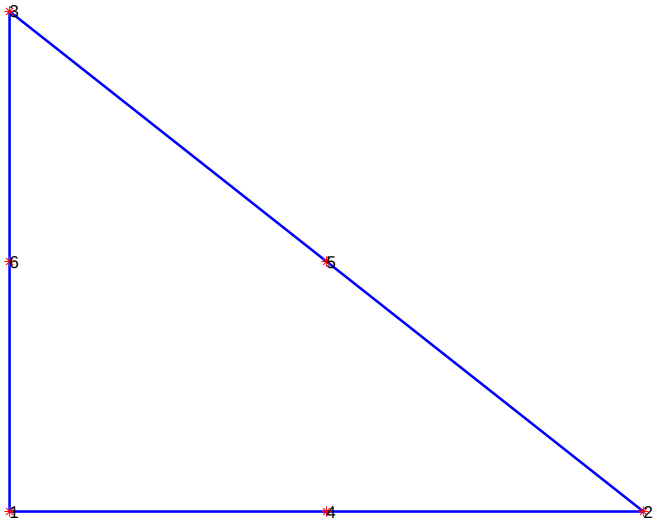
The first order partial derivatives of these shape function:

$$\begin{aligned}&\hat{\phi}_{1,1,\hat{x}}(\hat{x},\hat{y})=-1, \quad \hat{\phi}_{1,1,\hat{y}}(\hat{x},\hat{y})=-1\\ &\hat{\phi}_{2,1,\hat{x}}(\hat{x},\hat{y})=1, \quad \hat{\phi}_{2,1,\hat{y}}(\hat{x},\hat{y})=0\\ &\hat{\phi}_{3,1,\hat{x}}(\hat{x},\hat{y})=0, \quad \hat{\phi}_{3,1,\hat{y}}(\hat{x},\hat{y})=1\end{aligned}$$



For higher degree shape functions, we recall the Lemma 1.1: Let p be a polynomial of degree $k \geq 1$ that vanishes on a hyperplane described by linear polynomial $L(X)$. Then $p(X) = L(X)Q(X)$ where $deg(Q) \leq k-1$.

2nd degree shape functions: Let us consider $\hat{\phi}_{1,2}(\hat{x}, \hat{y}) \in \Pi_2$ associated with $\hat{A}_1$.



By definition,

$$\hat{\phi}_{1,2}(\hat{A}_1) = 1, \ \hat{\phi}_{1,2}(\hat{A}_i) = 0, \ 2 \leq i \leq 6$$

Since $\hat{\phi}_{1,2}(\hat{x}, \hat{y})$ is a polynomial of degree 2, so is its restriction on the line $\overline{\hat{A}_2\hat{A}_3}$. Since this quadratic polynomial is zero at three points $\hat{A}_2, \hat{A}_5, \hat{A}_3$ on $\overline{\hat{A}_2\hat{A}_3}$, $\hat{\phi}_{1,2}(\hat{x}, \hat{y})$ must be zero on the whole line determined by $\overline{\hat{A}_2\hat{A}_3}$. Hence, by Lemma 1.1,

$$\hat{\phi}_{1,2}(\hat{x}, \hat{y}) = \hat{\phi}_{1,1}(\hat{x}, \hat{y})Q(\hat{x}, \hat{y}),$$

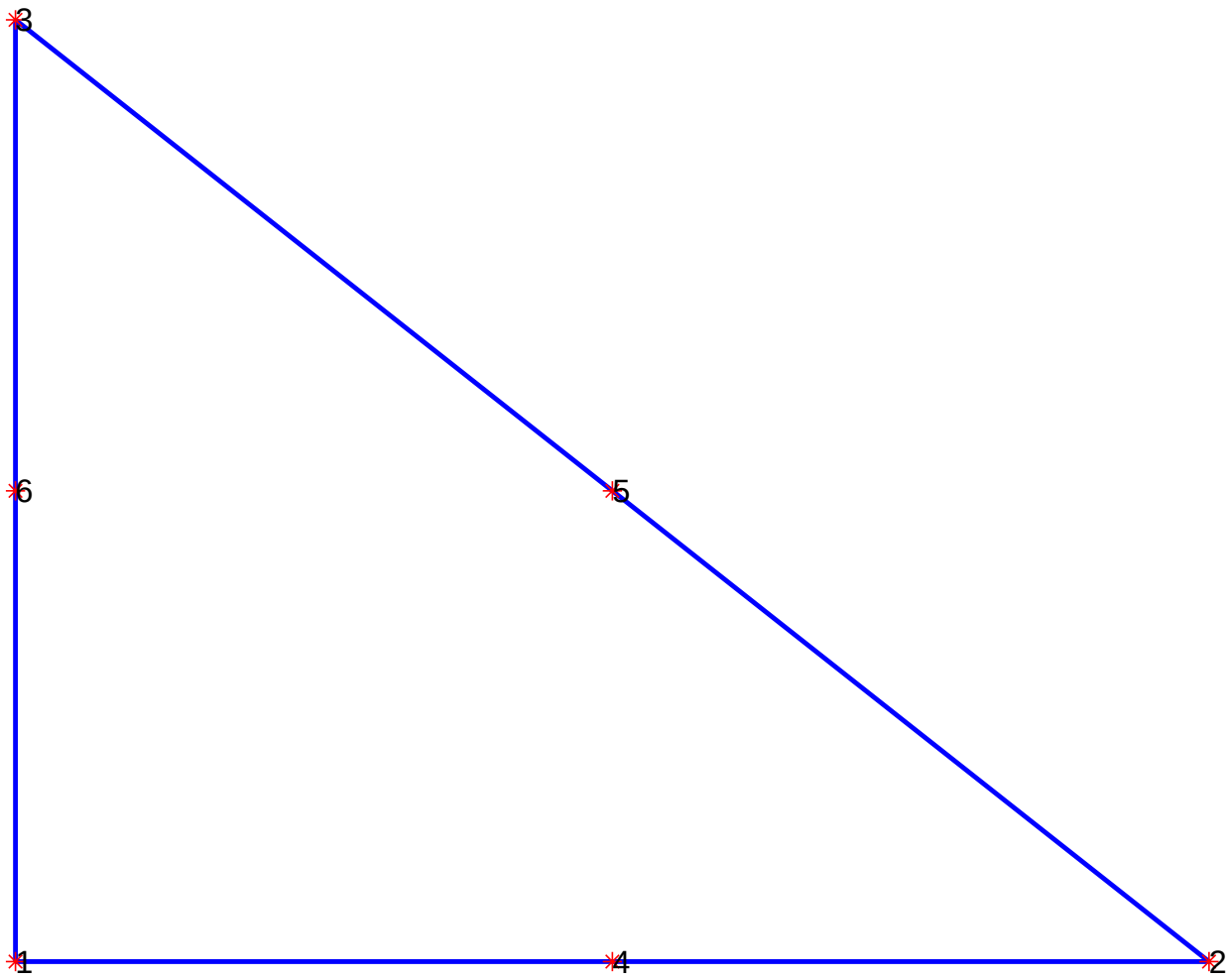
where $\hat{\phi}_{1,1}(\hat{x}, \hat{y})$ is the linear shape function associated with $\hat{A}_1$ and $Q(\hat{x}, \hat{y})$ is a polynomial of degree 1 or less.

Since $\hat{\phi}_{1,2}(\hat{A}_4) = \hat{\phi}_{1,2}(\hat{A}_6) = 0$ and , $\hat{\phi}_{1,1}(\hat{A}_4) \neq 0, \ \hat{\phi}_{1,1}(\hat{A}_6) \neq 0$, we must have $Q(\hat{A}_4) = Q(\hat{A}_6) = 0$. By Lemma 1.1 again, $Q(\hat{x}, \hat{y}) = c(\hat{\phi}_{1,1}(\hat{x}, \hat{y}) - 1/2)$. Hence $\hat{\phi}_{1,2}(\hat{x}, \hat{y}) = c\hat{\phi}_{1,1}(\hat{x}, \hat{y})(\hat{\phi}_{1,1}(\hat{x}, \hat{y}) - 1/2)$. Finally, from $\hat{\phi}_{1,2}(\hat{A}_1) = 1$, we have $c = 2$, and

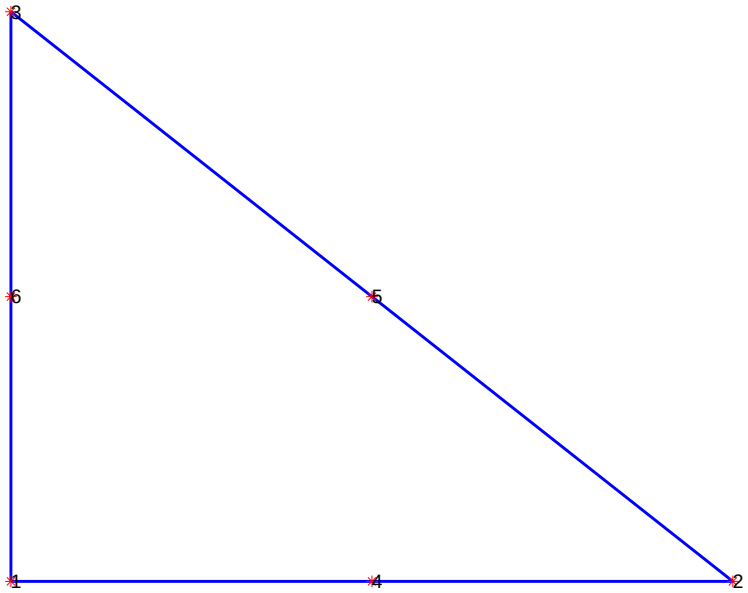
$$\hat{\phi}_{1,2}(\hat{x}, \hat{y}) = 2\hat{\phi}_{1,1}(\hat{x}, \hat{y})(\hat{\phi}_{1,1}(\hat{x}, \hat{y}) - 1/2)$$

Similarly, we can derive shape functions associated with the other two vertex nodes:

$$\hat{\phi}_{2,2}(\hat{x},\hat{y}) = 2\hat{\phi}_{2,1}(\hat{x},\hat{y})(\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/2) \in \Pi_2$$
$$\hat{\phi}_{3,2}(\hat{x},\hat{y}) = 2\hat{\phi}_{3,1}(\hat{x},\hat{y})(\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/2) \in \Pi_2$$



The same idea also applies to shape functions associated with those non-vertex nodes.



For $\hat{\phi}_{4,2}(\hat{x},\hat{y})$, we know that it must be zero at $\hat{A}_1, \hat{A}_2, \hat{A}_3, \hat{A}_5, \hat{A}_6$. Hence $\hat{\phi}_{4,2}(\hat{x},\hat{y})$ vanishes on two lines $\overline{\hat{A}_1\hat{A}_3}$ and $\overline{\hat{A}_2\hat{A}_3}$. By Lemma 1.1, we know that

$$\hat{\phi}_{4,2}(\hat{x},\hat{y}) = c\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})$$

Then applying the condition $\hat{\phi}_{4,2}(\hat{A}_4) = 1$, we have $c = 4$. Hence

$$\hat{\phi}_{4,2}(\hat{x},\hat{y}) = 4\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})$$

Similarly,

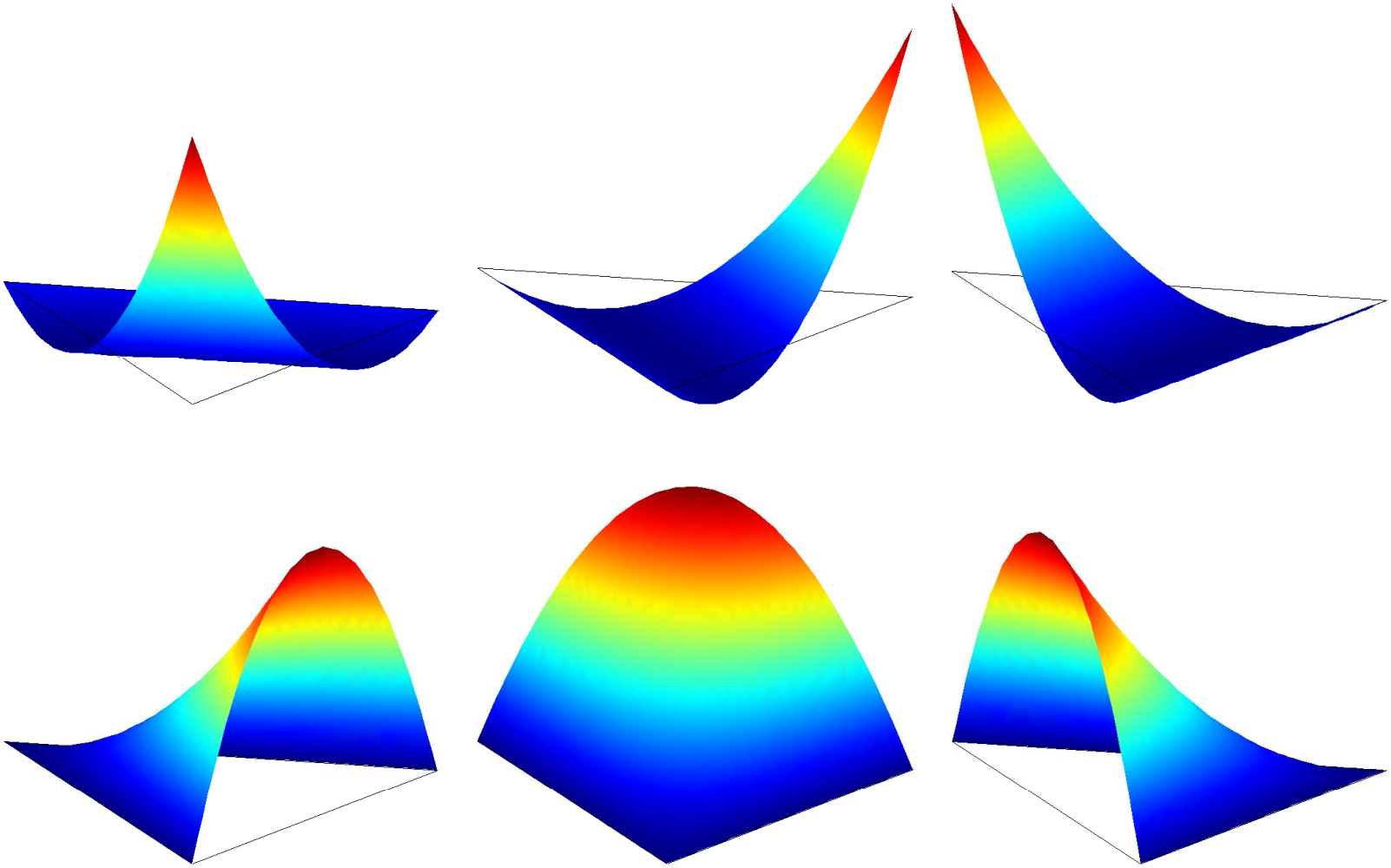
$$\hat{\phi}_{5,2}(\hat{x},\hat{y}) = 4\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})$$
$$\hat{\phi}_{6,2}(\hat{x},\hat{y}) = 4\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})$$

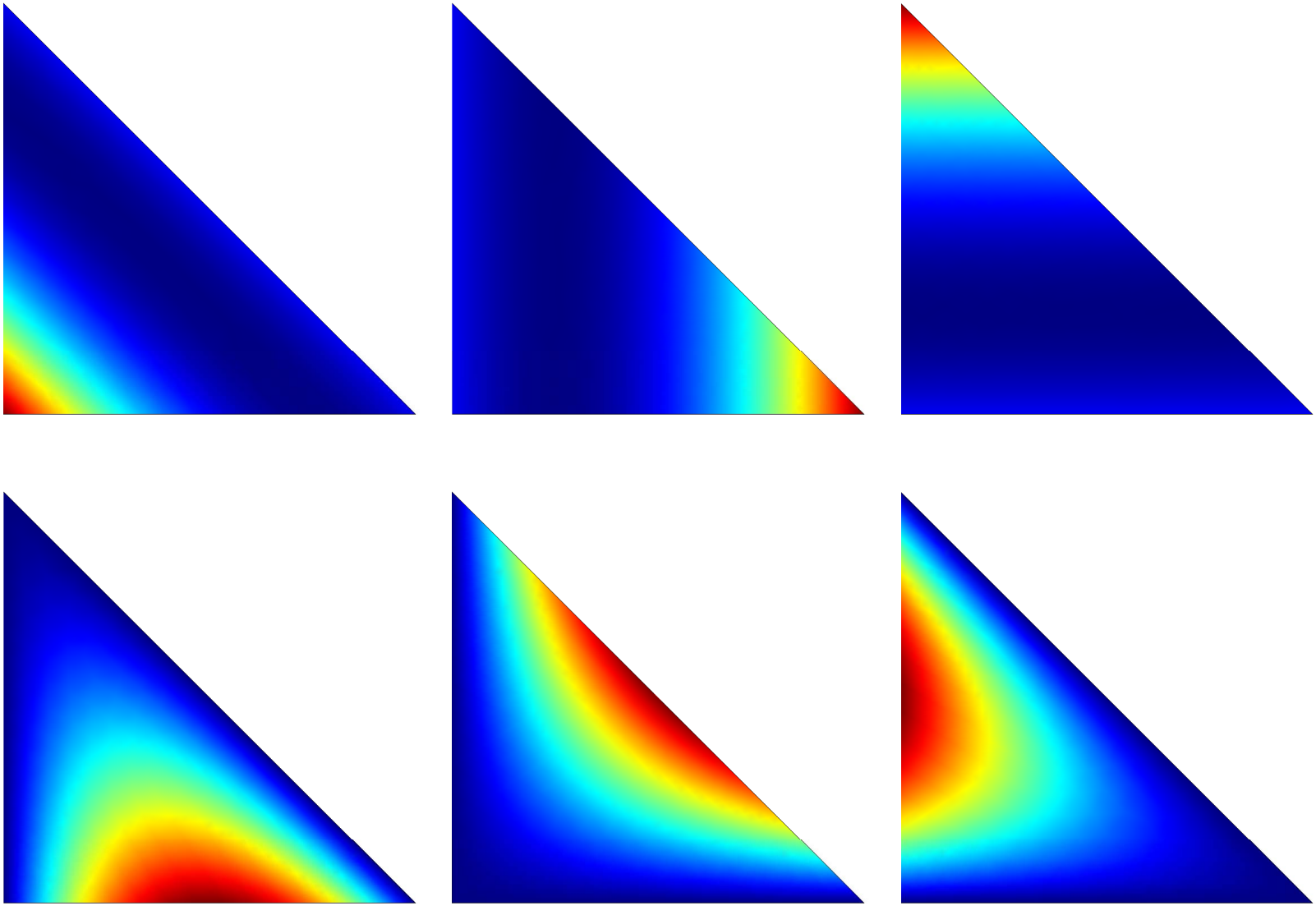
Formulas for the 2nd degree shape functions:

$$\begin{aligned}
\hat{\phi}_{1,2}(\hat{x},\hat{y}) &= 2\hat{\phi}_{1,1}(\hat{x},\hat{y})(\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/2)\\
\hat{\phi}_{2,2}(\hat{x},\hat{y}) &= 2\hat{\phi}_{2,1}(\hat{x},\hat{y})(\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/2)\\
\hat{\phi}_{3,2}(\hat{x},\hat{y}) &= 2\hat{\phi}_{3,1}(\hat{x},\hat{y})(\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/2)\\
\hat{\phi}_{4,2}(\hat{x},\hat{y}) &= 4\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\\
\hat{\phi}_{5,2}(\hat{x},\hat{y}) &= 4\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\\
\hat{\phi}_{6,2}(\hat{x},\hat{y}) &= 4\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})
\end{aligned}$$

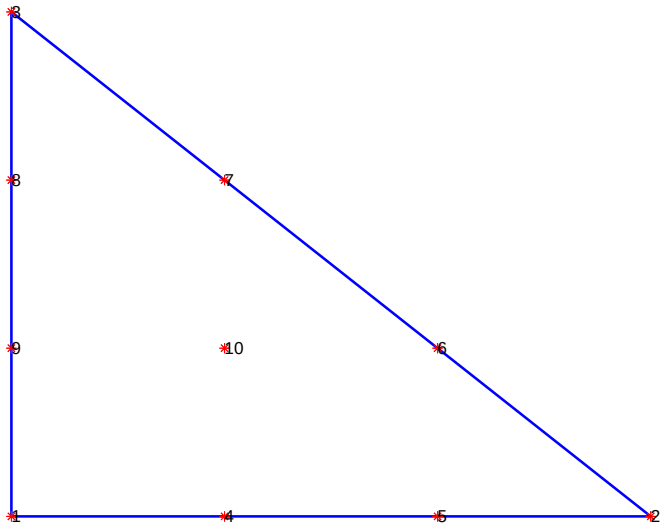
Their first order partial derivatives can be expressed in terms of first order partial derivatives of the 1st degree shape functions. For example

$$\begin{aligned}
\hat{\phi}_{1,2,\hat{x}}(\hat{x},\hat{y}) &= 2\hat{\phi}_{1,1,\hat{x}}(\hat{x},\hat{y})(\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/2) + 2\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{1,1,\hat{x}}(\hat{x},\hat{y})\\
\hat{\phi}_{4,2,\hat{y}}(\hat{x},\hat{y}) &= 4\hat{\phi}_{1,1,\hat{y}}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y}) + 4\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1,\hat{y}}(\hat{x},\hat{y})
\end{aligned}$$





3rd degree shape functions: For the shape function $\hat{\phi}_{1,3}(\hat{x},\hat{y})$ associated with the vertex node $\hat{A}_1$, let us first recall that the Lagrange nodes on $\hat{K}$ are ordered as follows:



Since $\hat{\phi}_{1,3}(\hat{x},\hat{y})$ vanishes on the line $\overline{\hat{A}_2\hat{A}_3}$, it should have $\hat{\phi}_{1,1}(\hat{x},\hat{y})$ as a factor according to Lemma 1.1. Since its values are zero at $\hat{A}_5, \hat{A}_8, \hat{A}_{10}$, it contains $\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3$ as a factor. Similarly, by the condition that $\hat{\phi}_{1,3}(\hat{x},\hat{y})$ is zero at $\hat{A}_4, \hat{A}_9$, it contains $\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 2/3$ as a factor. Hence,

$$\hat{\phi}_{1,3}(\hat{x},\hat{y}) = c\hat{\phi}_{1,1}(\hat{x},\hat{y})\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\big]\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 2/3\big]$$

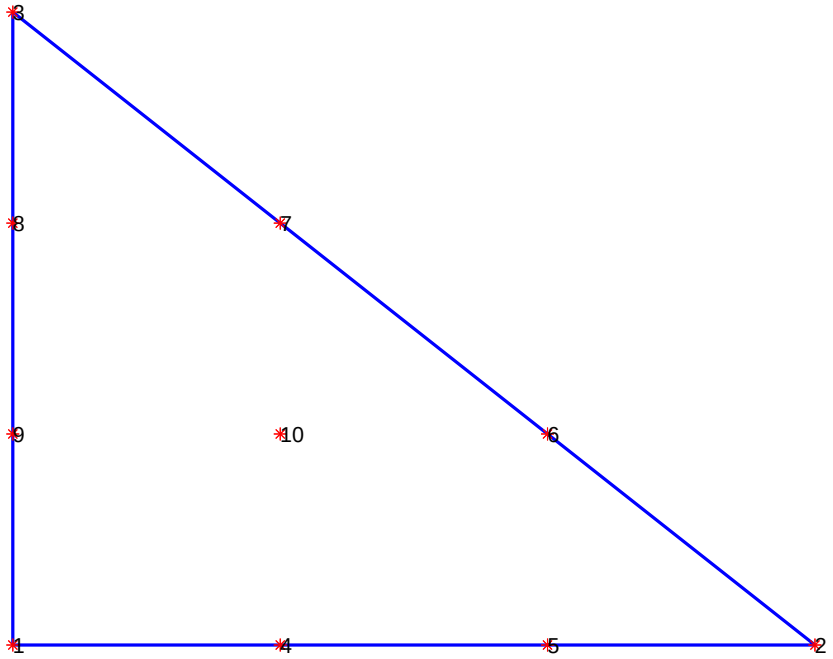
Finally, by the condition that $\hat{\phi}_{1,3}(\hat{A}_1) = 1$, we have $c = \frac{9}{2}$. Therefore,

$$\hat{\phi}_{1,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\big]\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 2/3\big]$$

Similarly, we show that

$$\hat{\phi}_{2,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\right]\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 2/3\right]$$

$$\hat{\phi}_{3,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\right]\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 2/3\right]$$



We now consider the shape function $\hat{\phi}_{4,3}(\hat{x},\hat{y})$ associated with the edge node $\hat{A}_4$. Since $\hat{\phi}_{4,3}(\hat{x},\hat{y})$ vanishes on the two lines $\overline{\hat{A}_2\hat{A}_3}$ and $\overline{\hat{A}_3\hat{A}_1}$, by Lemma 1.1, it should contain $\hat{\phi}_{1,1}(\hat{x},\hat{y})$ and $\hat{\phi}_{2,1}(\hat{x},\hat{y})$, and this has used all the degrees of freedom except for $\hat{A}_4$ and $\hat{A}_5$. Since $\hat{\phi}_{4,3}(\hat{x},\hat{y})$ vanishes at $\hat{A}_5$, it contains $\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3$ as a factor. finally, by $\hat{\phi}_{4,3}(\hat{A}_4) = 1$, we have

$$\hat{\phi}_{4,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\big]$$

Similarly, we can derive the following formulas for 3-rd degree Lagrange shape functions associated with other edge nodes:

$$\hat{\phi}_{5,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\big[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\big]$$

$$\hat{\phi}_{6,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})\big[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\big]$$

$$\hat{\phi}_{7,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})\big[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\big]$$

$$\hat{\phi}_{8,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})\big[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\big]$$

$$\hat{\phi}_{9,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})\big[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\big]$$

Using the same arguments, we obtain the following formula for $\hat{\phi}_{10,3}(\hat{x},\hat{y})$:

$$\hat{\phi}_{10,3}(\hat{x},\hat{y}) = 27\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})$$

Formulas for 3rd degree shape functions:

$$\hat{\phi}_{1,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\left[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\right]\left[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 2/3\right]$$

$$\hat{\phi}_{2,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\right]\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 2/3\right]$$

$$\hat{\phi}_{3,3}(\hat{x},\hat{y}) = \frac{9}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\right]\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 2/3\right]$$

$$\hat{\phi}_{4,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\left[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{5,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{6,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})\left[\hat{\phi}_{2,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{7,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{8,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})\left[\hat{\phi}_{3,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{9,3}(\hat{x},\hat{y}) = \frac{27}{2}\hat{\phi}_{3,1}(\hat{x},\hat{y})\hat{\phi}_{1,1}(\hat{x},\hat{y})\left[\hat{\phi}_{1,1}(\hat{x},\hat{y}) - 1/3\right]$$

$$\hat{\phi}_{10,3}(\hat{x},\hat{y}) = 27\hat{\phi}_{1,1}(\hat{x},\hat{y})\hat{\phi}_{2,1}(\hat{x},\hat{y})\hat{\phi}_{3,1}(\hat{x},\hat{y})$$

Remark: First order partial derivatives of these 3rd degree shape functions can be expressed in terms of first order partial derivatives of the 1st degree shape functions.

Remark: Higher degree Lagrange shape functions can be implemented with the code for the linear Lagrange shape functions.

For each $i = 1, 2, \cdots, (p+1)(p+2)/2$, we call the p-th degree polynomial $\hat{\phi}_{i,p}(\hat{X}) = \hat{\phi}_{i,p}(\hat{x}, \hat{y}) \in V^p(\hat{K})$ such that

$$\hat{\phi}_{i,p}(\hat{A}_j) = \delta_{ij}, \quad j = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}$$

a p-th degree Lagrange shape function for $V^p(\hat{K})$ associated with the i-th Lagrange node $\hat{A}_i$ in the reference element $\hat{K}$, and it is easy show that there shape functions are linearly independent ???

The p-th degree finite element space on $\hat{K}$:

$$V^p(\hat{K}) = span\left\{\hat{\phi}_{i,p}, \ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}\right\} = \Pi_p \quad ???$$

How to implement the local shape functions $\hat{\phi}_{i,p}(\hat{x}, \hat{y})$ on $\hat{K}$:

```
function phih = shape_fun_2D_Lagrange_tri_ref(hx, hy, degree, ...
                    shape_index, d_hx, d_hy)
```

where `d_hx, d_hy = 0 or 1` specify the derivatives for the shape function

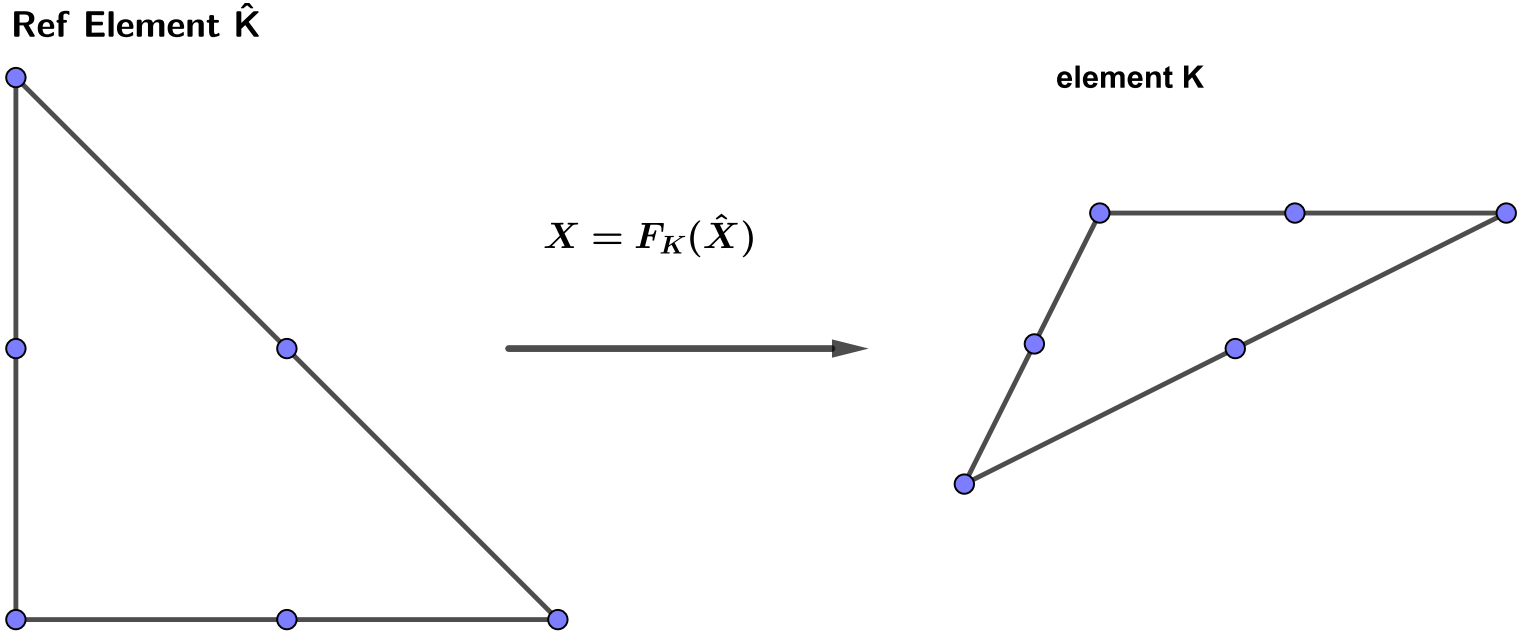
Remark: This program will be given in the next Homework.

Finite element nodes on each element of a mesh: For $K = \triangle A_1A_2A_3 \in \mathcal{T}_h$, we introduce $(p+1)(p+2)/2$ finite element nodes in K:

$$A_i = F_K(\hat{A}_i) = (A_3 - A_1, A_2 - A_1)\hat{A}_i + A_1, \quad i = 1, 2, \cdots, (p+1)(p+2)/2$$

where $\hat{A}_i, i = 1, 2, \cdots, (p+1)(p+2)/2$ are the p-th degree Lagrange nodes on the reference element.

Quadratic nodes on an element K:



The local p-th degree finite element space on $K = \triangle A_1A_2A_3 \in \mathcal{T}_h$: Then, for each finite element node $A_i \in K$, we define its associate p-th degree shape function by

$$\phi_{i,p,K}(x,y) = \hat{\phi}_{i,p}(\hat{x}(x,y), \hat{y}(x,y)), \quad i = 1, 2, \cdots, (p+1)(p+2)/2$$

$$\text{where} \quad B = (A_2 - A_1, A_3 - A_1) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}, \quad \hat{B} = B^{-1} = \begin{pmatrix} \hat{b}_{11} & \hat{b}_{12} \\ \hat{b}_{21} & \hat{b}_{22} \end{pmatrix}$$

$$\hat{x} = \hat{x}(x,y) = \hat{b}_{11}(x - x_1) + \hat{b}_{12}(y - y_1)$$

$$\hat{y} = \hat{y}(x,y) = \hat{b}_{21}(x - x_1) + \hat{b}_{22}(y - y_1)$$

Then, we define the local Lagrange type p-th degree finite element space on element $K \in \mathcal{T}_h$ as follows

$$V_h^p(K) = span\left\{ \phi_{i,p,K}, \ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2} \right\}$$

Implementation of local shape functions $\phi_{i,p,K}(x,y)$ on an element $K = \triangle A_1A_2A_3 \in \mathcal{T}_h$:

$$\text{recall that} \quad B = (A_2 - A_1, A_3 - A_1) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}, \quad \hat{B} = B^{-1} = \begin{pmatrix} \hat{b}_{11} & \hat{b}_{12} \\ \hat{b}_{21} & \hat{b}_{22} \end{pmatrix}$$

$$\hat{x} = \hat{x}(x,y) = \hat{b}_{11}(x - x_1) + \hat{b}_{12}(y - y_1)$$

$$\hat{y} = \hat{y}(x,y) = \hat{b}_{21}(x - x_1) + \hat{b}_{22}(y - y_1)$$

$$\phi(x,y) = \hat{\phi}(\hat{x}(x,y), \hat{y}(x,y)) \tag{5}$$

$$\phi_x(x,y) = \hat{b}_{11}\hat{\phi}_{\hat{x}}(\hat{x}(x,y), \hat{y}(x,y)) + \hat{b}_{21}\hat{\phi}_{\hat{y}}(\hat{x}(x,y), \hat{y}(x,y)) \tag{6}$$

$$\phi_y(x,y) = \hat{b}_{12}\hat{\phi}_{\hat{x}}(\hat{x}(x,y), \hat{y}(x,y)) + \hat{b}_{22}\hat{\phi}_{\hat{y}}(\hat{x}(x,y), \hat{y}(x,y)) \tag{7}$$

The Matlab function for shape function $\phi_{i,p,K}(x,y)$ on $K \in \mathcal{T}_h$:

```
function phi = shape_fun_2D_Lagrange_tri(x, y, elem, degree, ...
                    shape_index, d_x, d_y)
B = [elem(:, 2) - elem(:, 1), elem(:, 3) - elem(:, 1)]; Bh = inv(B);
hx = Bh(1,1)*(x - elem(1,1)) + Bh(1,2)*(y - elem(2,1));
hy = Bh(2,1)*(x - elem(1,1)) + Bh(2,2)*(y - elem(2,1));
if d_x = 0 and d_y = 0: use (5) for phi
if d_x = 1 and d_y = 0: use (6) for phi
if d_x = 0 and d_y = 1: use (7) for phi
```

Here, `elem` is a 2×3 array defining the 3 vertices of a triangular element K. We often generate `elem` by `elem = mesh.p(:, mesh.t(:,k))`.

Example: Here is a script for using this function:

```
elem = [0, 0.1, 0.05;1, 1.1, 1.2]; % the element
x = [0.015192976098518 0.084807023901482]; % x-coordinates
y = [1.030385952197037 1.100000000000000]; % y-coordinates
degree = 1;
shape_index = 1;
d_x = 0; d_y = 0;
f = shape_fun_2D_Lagrange_tri(x, y, elem, degree, ...
                                  shape_index, d_x, d_y);
d_x = 1; d_y = 0;
fx = shape_fun_2D_Lagrange_tri(x, y, elem, degree, ...
                                  shape_index, d_x, d_y);
d_x = 0; d_y = 1;
fy = shape_fun_2D_Lagrange_tri(x, y, elem, degree, ...
                                  shape_index, d_x, d_y);
fprintf('f  = %.15e, %.15e\n', f(1), f(2))
fprintf('fx = %.15e, %.15e\n', fx(1), fx(2))
fprintf('fy = %.15e, %.15e\n', fy(1), fy(2))
```

which produces the following results:

```
f  = 7.974269853530896e-01, 1.012865073234533e-01
fx = -6.666666666666662e+00, -6.666666666666662e+00
fy = -3.333333333333334e+00, -3.333333333333334e+00
```

More data are given in data_for_debugging_2D_ShapeFunctions.txt on Canvas for testing this Matlab function.

3.2.2, 2D C^0 p-th degree FE space on Ω:

- Form a mesh $\mathcal{T}_h$ of Ω.
- Form the set of finite element nodes for the p-th degree finite element space by collecting the p-th degree Lagrange nodes from all elements of $\mathcal{T}_h$, denote this set by $\mathcal{N}_{h,dof}$. We index these finite element nodes.
- Then, put indices of finite element nodes in the connectivity matrix/array T_{dof} of size $((p+1)(p+2)/2) \times |\mathcal{T}_h|$ such that its j-th column contains indices of the finite element nodes in the j-th element ordered according to the p-th degree Lagrange nodes locally on that element.
- For each $X_j = (x_j, y_j) \in \mathcal{N}_{h,dof}$, we define a piecewise polynomial $\phi_j(X)$ such that:

$$\phi_j|_K \in V_h^p(K), \ \forall K \in \mathcal{T}_h \tag{8}$$

$$\phi_j(X_i) = \delta_{ij}, \ \forall X_i \in \mathcal{N}_{h,dof} \qquad \text{(Lagrange property)} \tag{9}$$

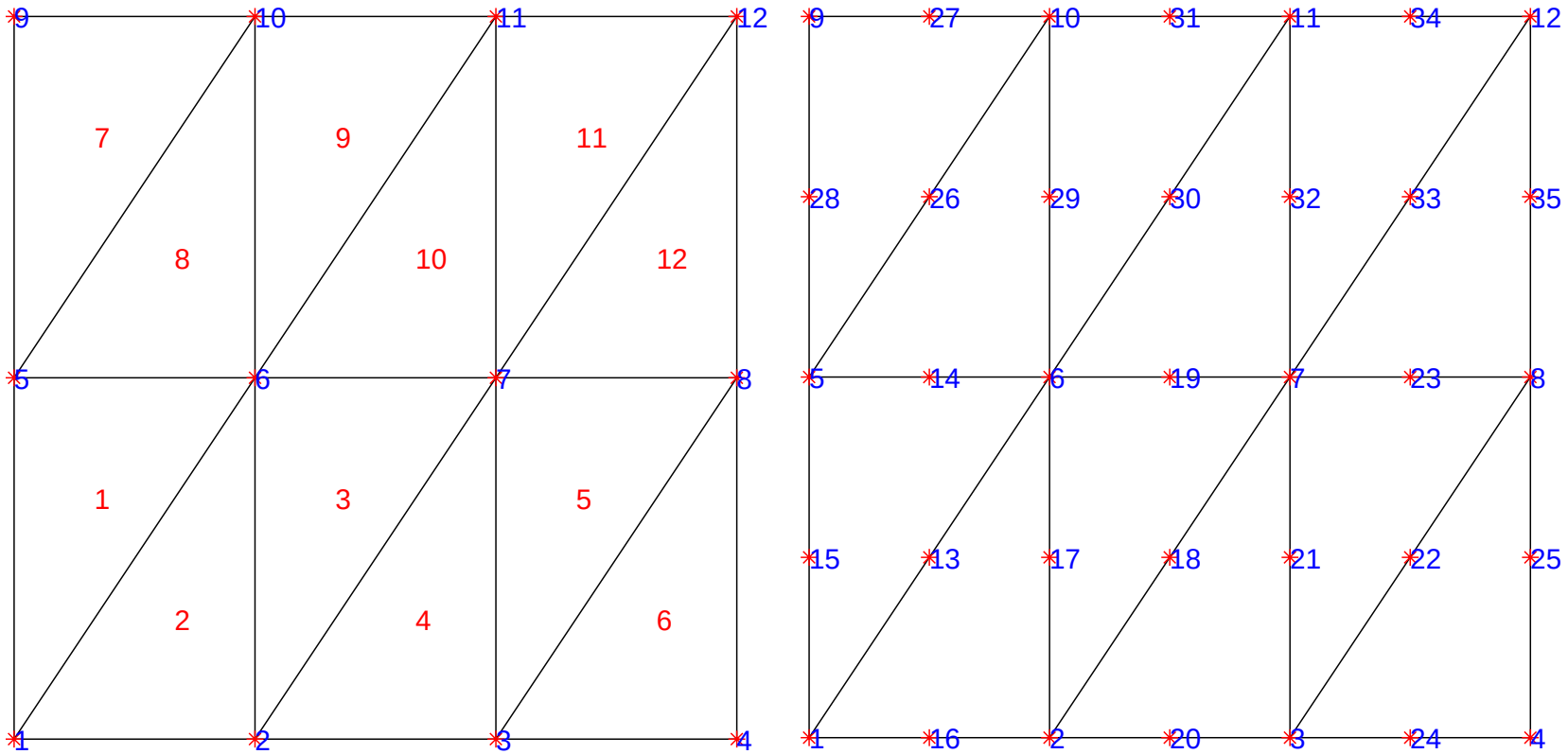
  Each $\phi_j, \ 1 \leq j \leq |\mathcal{N}_{h,dof}|$ is called a global basis function, and, because of the Lagrange nodes, we can show that $\phi_j(X)$ is a continuous function ???

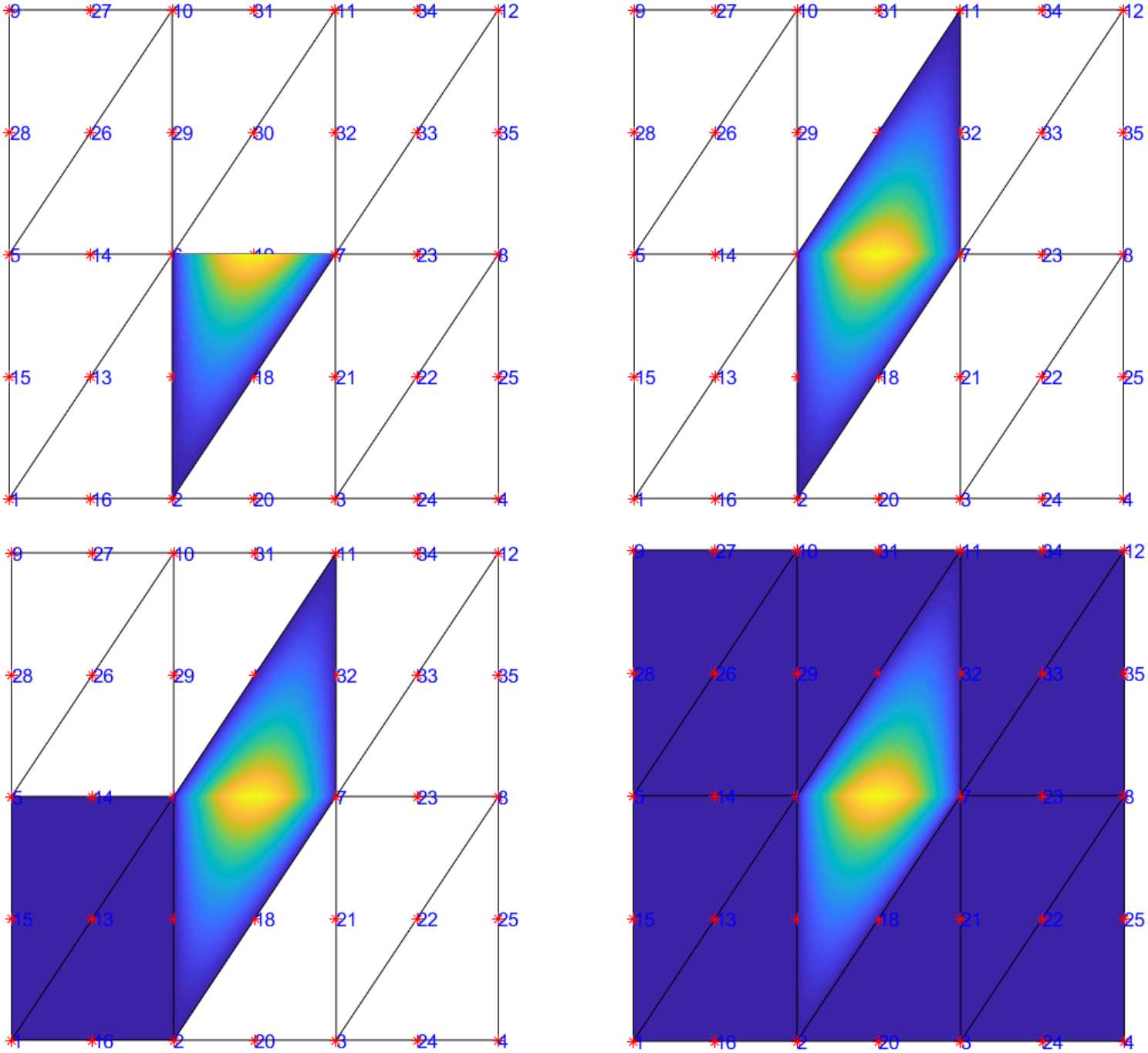
- The C^0 p-th degree Lagrange type FE space is defined as

$$V_h^p(\Omega) = span\{\phi_i, \ \forall X_i \in \mathcal{N}_{h,dof}\} \tag{10}$$

Some of the basis functions in $V_h^2(\Omega)$ which is prepared as follows:

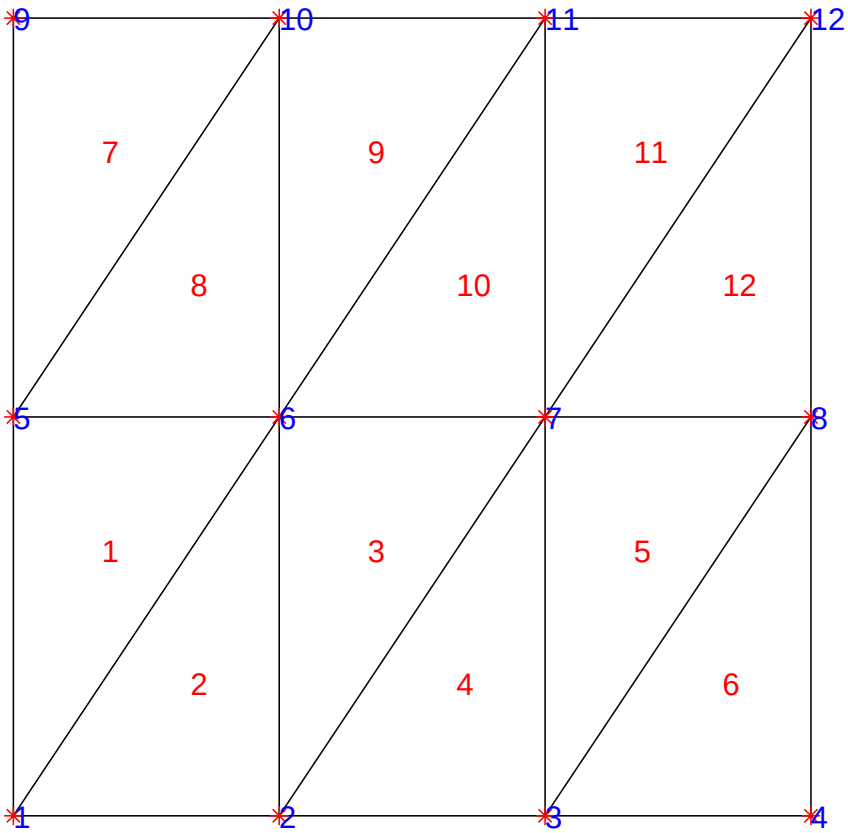
```
xmin = 0; xmax = 5; ymin = 0; ymax = 5;
nx = 3; ny = 2;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny);
figure(1); clf; FE_mesh_viewer_2D(mesh, mesh.p, 1, 1)
C = 1; B = 1; [e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne; degree = 2;
fem = fem_generator_Lagrange_2D_tri_general(mesh, degree);
figure(2); clf; FE_mesh_viewer_2D(mesh, fem.p, 0, 1)
```

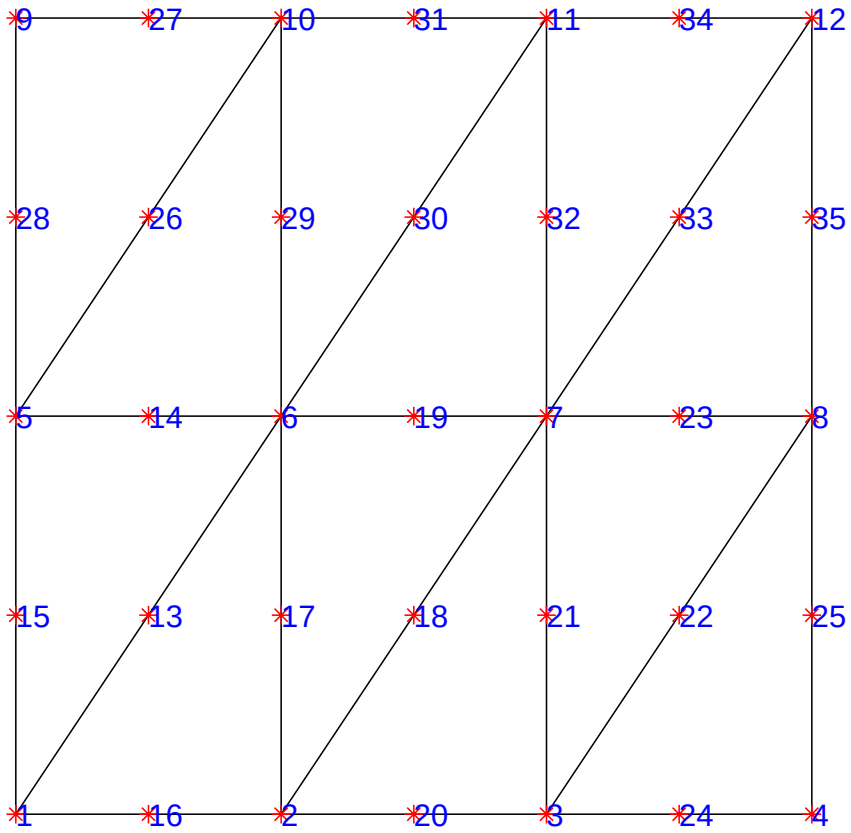


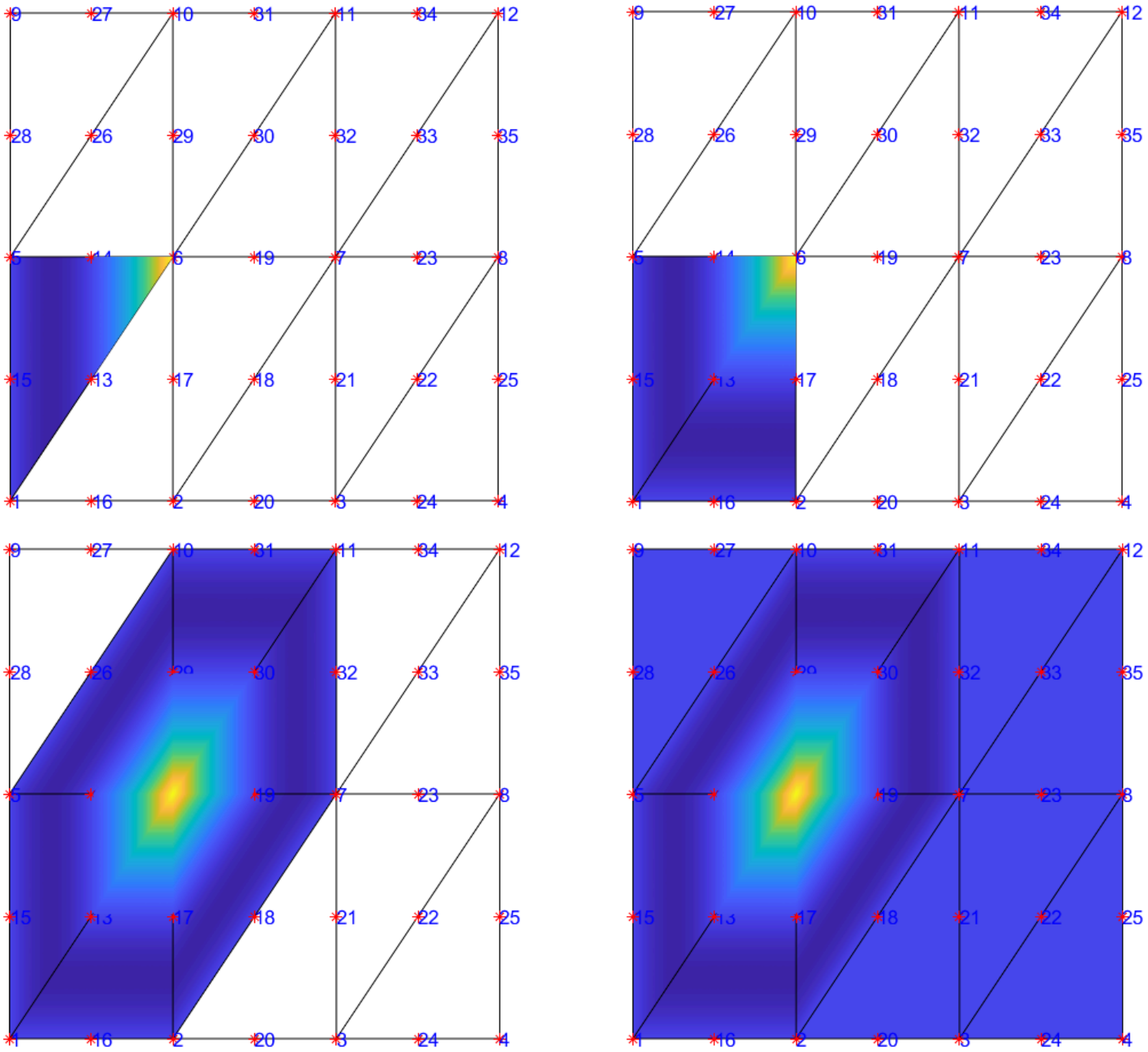


The shape function displayed in Matlab: `openfig('qadratic_basis_19_5')`

Recall the `mesh` and `fem`:







The shape function displayed in Matlab: `openfig('qadratic_basis_6_9')`

Properties of the finite element basis functions:

- We can verify that the basis functions $\phi_i, i = 1, 2, \cdots, |\mathcal{N}_{h,dof}|$ spanning $V_h^p(\Omega)$ is continuous ??? Hence, by Theorem **??**, we know $\phi_i \in H^1(\Omega)$ and

$$V_h^p(\Omega) = span\{\phi_i,\ \forall X_i \in \mathcal{N}_{h,dof}\} \subset H^1(\Omega)$$

- These finite element basis functions are linearly independent according to the property specified in (9) (???), and $dim(V_h^p(\Omega)) = |\mathcal{N}_{h,dof}|$.
- Every finite element function u_h in $V_h^p(\Omega)$ can be written as

$$u_h(X) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X) \tag{11}$$

which means a finite element function $u_h(X)$ is determined by the array $\vec{u} = (u_1, u_2, \cdots, u_{|\mathcal{N}_{h,dof}|})^t$. We note the following Lagrange property:

$$u_h(X_i) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X_i) = u_i, \quad i = 1, 2, \cdots, |\mathcal{N}_{h,dof}| \tag{12}$$

Hence the coefficient u_i in a finite element function $u_h(X) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X)$ is its value at the global node $X_i \in \mathcal{N}_{h,dof}$.

3.2.3, Implementation of global finite element basis functions:

As before, we emphasize again that finite element functions are rarely directly used globally. Computations involving finite element functions are generally carried out element by element without direct involvement of the global finite element basis functions. In general, we even do not have to program the global basis functions.

The following arrays are key components for implementing this idea for the finite element space $V_h^p(\Omega)$.

- For $\mathcal{N}_{h,dof}$: an array `p` containing coordinates for all finite element nodes of the finite element space. It can be verified that

$$\text{the number of FE nodes:} \qquad |\mathcal{N}_{h,dof}| = \begin{cases} |\mathcal{N}_h|, & \text{when } p = 1 \\ |\mathcal{N}_h| + |\mathcal{E}_h|, & \text{when } p = 2 \\ |\mathcal{N}_h| + 2|\mathcal{E}_h| + |\mathcal{T}_h|, & \text{when } p = 3 \\ \vdots \end{cases}$$

  The i-th column of p contains the x-y coordinates of the i-th finite element node. The key idea is to count the nodes in the interior of each edge and each element.

- For T_{dof}: the connectivity matrix `t` for the global degrees of freedom. The k-th column of t contains the indices of the finite element nodes inside k-th element.

We can implement these matrices with a Matlab function as follows:

```
function fem = fem_generator_Lagrange_2D_tri(mesh, degree)
```

`fem` is a structure for describing finite element basis functions.

`fem.p` provides coordinates for points in $\mathcal{N}_{h,dof}$.

`fem.t` is the connectivity matrix T_{dof} for the global degrees of freedom.

`fem.degree` is the degree of polynomials used for the finite element space.

In principle, both `fem.p` and `fem.t` can be prepared element-by-element over a given a basic mesh. But, here we assume that the data structure `mesh` has been prepared with the following fields: `mesh.p`, `mesh.t`, `e`, and `M_ne`.

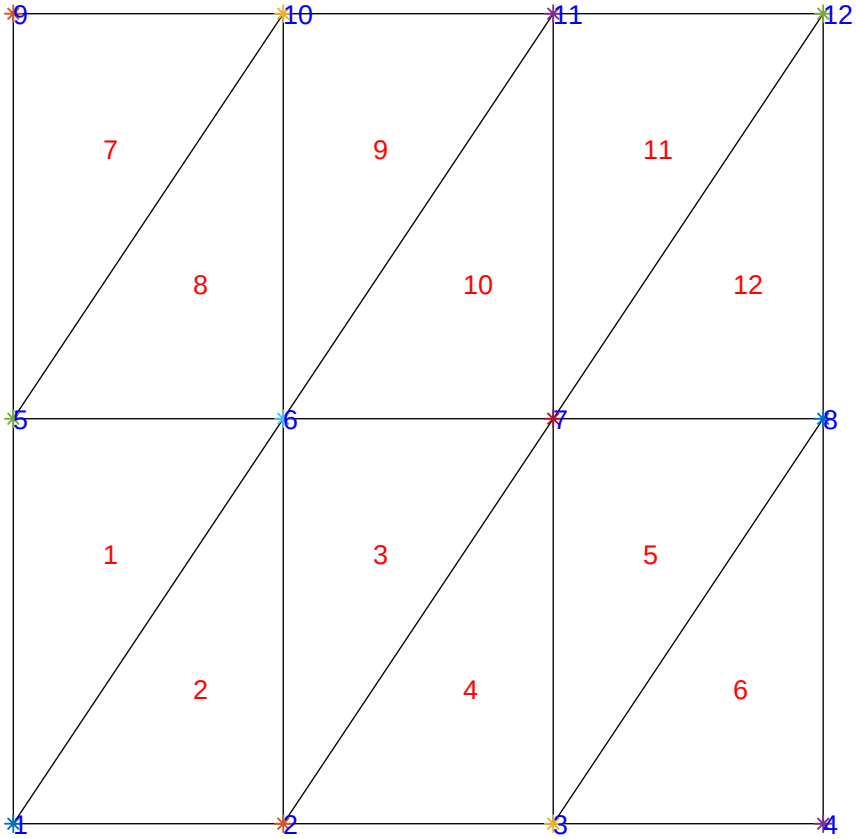
For $V_h^1(\Omega)$: For `degree=1`, `fem.p` and `fem.t` can be completely determined by `mesh` as follows.

```
p = mesh.p; t = mesh.t;
fem = struct('p', p, 't', t, 'degree', degree);
```

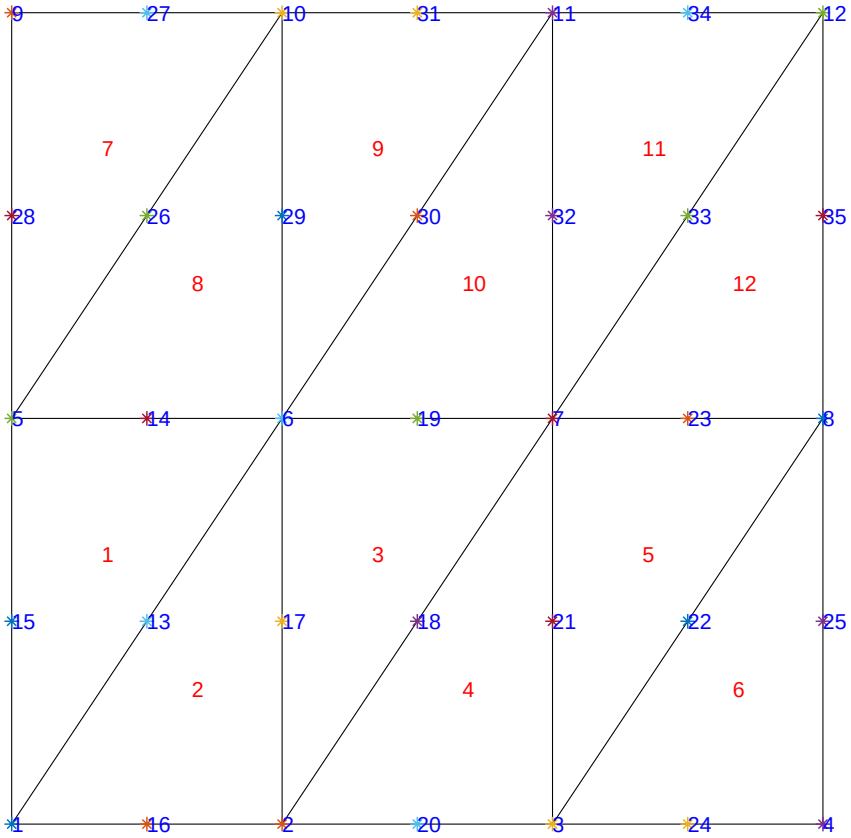
There is no need to use the arrays `e` and `M_ne`

How to prepare `fem` for higher degree finite element function:

Mesh nodes:

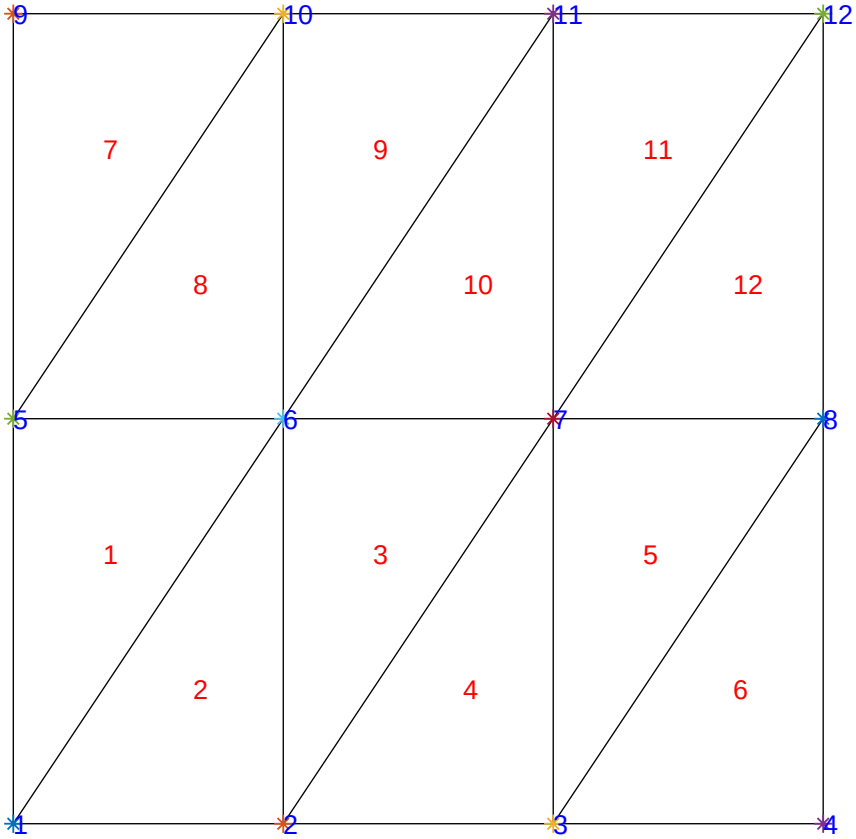


FEM nodes for $V_h^2(\Omega)$:

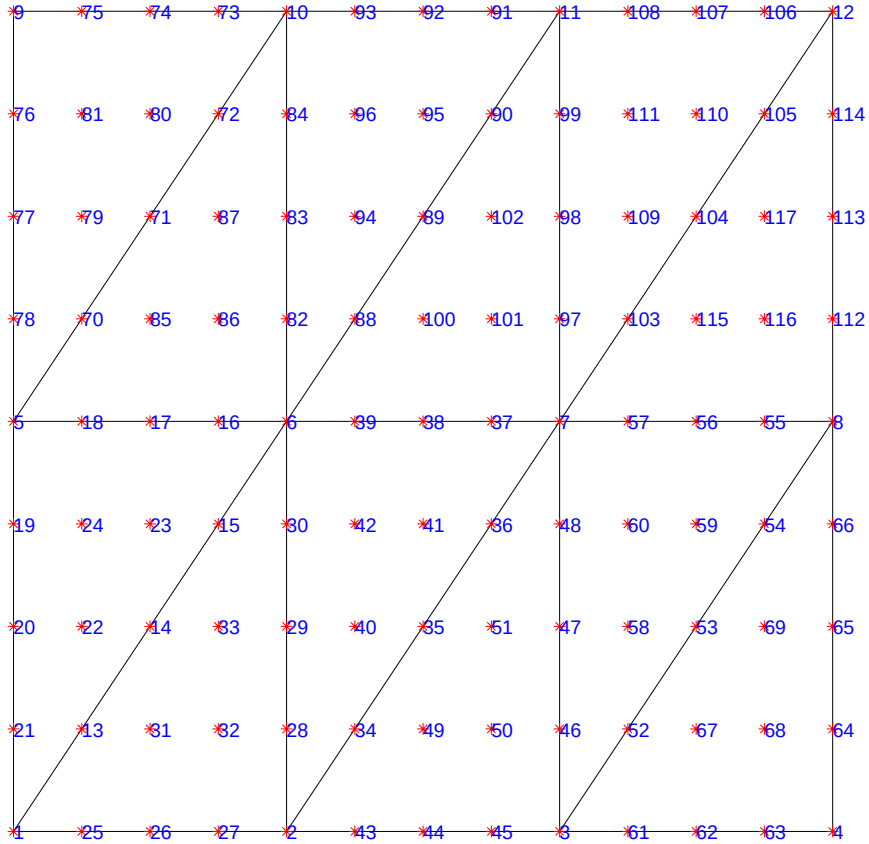


How to prepare `fem` for higher degree finite element function:

Mesh nodes:



FEM nodes for $V_h^4(\Omega)$:



For $V_h^2(\Omega)$: For `degree=2`, for preparation, we let `e_ind = zeros(1, length(mesh.e));`. This array `e_ind` is for the bookkeeping about which edge has been treated for its interior finite element nodes in the procedure below.

We first organize those finite element nodes forming vertices of elements by initializing `p` and `t` as follows

```
p = zeros(2, length(mesh.p) + length(e));
p(:, 1:length(mesh.p)) = mesh.p; % all the mesh nodes are FE nodes
t = zeros(6, length(mesh.t)); % each element has 6 nodes
t(1:3, :) = mesh.t; % the first 3 nodes are mesh nodes
p_count = length(mesh.p);
```

We can then add finite element nodes in the interior of edges to `p` and `t` element-by-element.

On the k-th element, assume its three edges are formed by its vertices 1-2, 2-3, and 3-1, respectively. We can extract the coordinates of all the vertices by `vert = mesh.p(:, t(:, k))`.

By the definition of `M_ne`, we know that edge 1 of the k-th element is the `M_ne(t(1, k), t(2, k))`-th edge in the mesh.
If `e_ind(M_ne(t(1, k), t(2, k)))` is zero, then this edge is not treated yet; hence, we count the middle point of this edge as a new finite element node by

```
p_count = p_count + 1;
p(:, p_count) = (1/2)*vert(:, i) + (1/2)*vert(:, j);
e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) = p_count;
t(4, k) = p_count;
```

Otherwise, edge 1 has been treated, there is no need to count its middle point as a new finite element node anymore and we just let

```
t(4, k) = e_ind(M_ne(mesh.t(i, k), mesh.t(j, k)));
```

The same procedure can be repeated on edge 2 and edge 3. Then, we can repeat on the next element and so on.

Below is a Matlab script demonstrating how to constructing the p and t arrays for $V_h^2(\Omega)$ with a given mesh.

```
M_ne = mesh.M_ne;
p = zeros(2, length(mesh.p) + length(mesh.e));
p(:, 1:length(mesh.p)) = mesh.p; % FE nodes at vertices
t = zeros(6, length(mesh.t)); t(1:3, :) = mesh.t;
p_count = length(mesh.p);  e_ind = zeros(1, length(mesh.e));

% FE nodes inside edges, not end points of edges
for k = 1:length(mesh.t)
    vert = mesh.p(:, mesh.t(:, k));

    i = 1; j = 2; % deal with the 1st edge of the k-th elem
    if e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) == 0 % not treated
        p_count = p_count + 1;
        p(:, p_count) = (1/2)*vert(:, i) + (1/2)*vert(:, j);
        e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) = p_count;
        t(4, k) = p_count;
    else
        t(4, k) = e_ind(M_ne(mesh.t(i, k), mesh.t(j, k)));
    end
```

```
    i = 2; j = 3; % deal with the 2nd edge of the k-th elem
    if e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) == 0 % not treated
        p_count = p_count + 1;
        p(:, p_count) = (1/2)*vert(:, i) + (1/2)*vert(:, j);
        e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) = p_count;
        t(5, k) = p_count;
    else
        t(5, k) = e_ind(M_ne(mesh.t(i, k), mesh.t(j, k)));
    end

    i = 3; j = 1; % deal with the 3rd edge of the k-th elem
    if e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) == 0 % not treated
        p_count = p_count + 1;
        p(:, p_count) = (1/2)*vert(:, i) + (1/2)*vert(:, j);
        e_ind(M_ne(mesh.t(i, k), mesh.t(j, k))) = p_count;
        t(6, k) = p_count;
    else
        t(6, k) = e_ind(M_ne(mesh.t(i, k), mesh.t(j, k)));
    end
end
fem = struct('p', p, 't', t, 'degree', degree);
```

Remark: This script can be readily extended for $V_h^p(\Omega)$ with $p > 2$.

A general template for `fem_generator_Lagrange_2D_tri`:

```
function fem = fem_generator_Lagrange_2D_tri_general(mesh, degree)
M_ne = mesh.M_ne;
if degree == 1
    p = mesh.p; t = mesh.t;
elseif degree >= 2
    ???
end
```

Here is a script for how to use `fem_generator_Lagrange_2D_tri`:

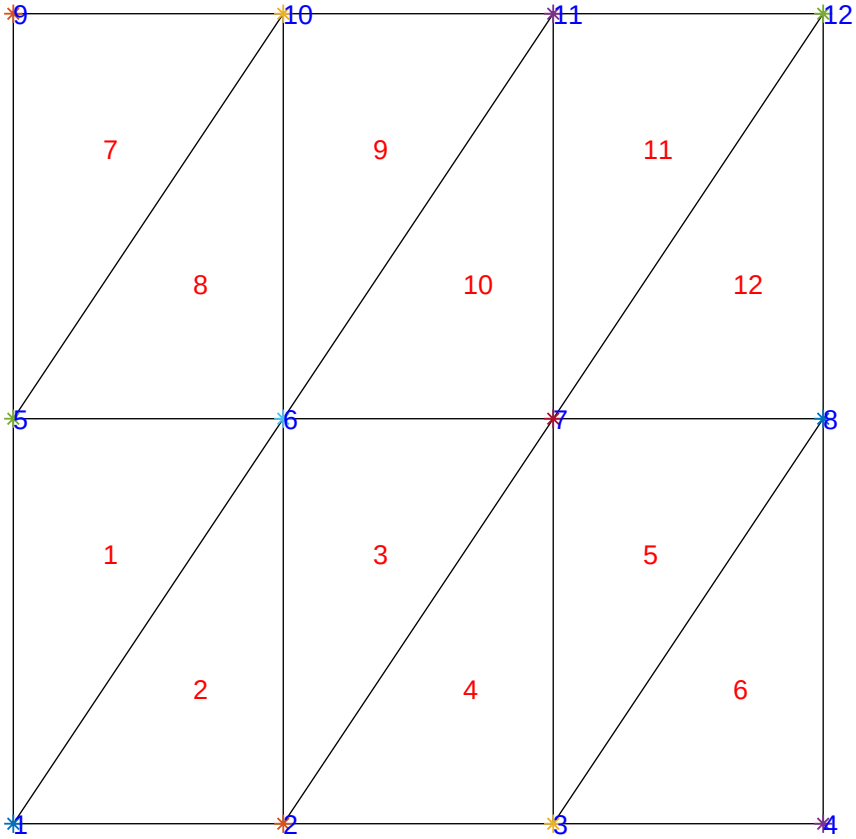
```
xmin = 0; xmax = 1; ymin = 0; ymax = 1;
nx = 3; ny = 2;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny);
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne; % enrich the mesh
degree = 1;
fem = fem_generator_Lagrange_2D_tri(mesh, degree);
```

We can see that `mesh.p = fem.p`, `mesh.t = fem.t` because `degree = 1`.

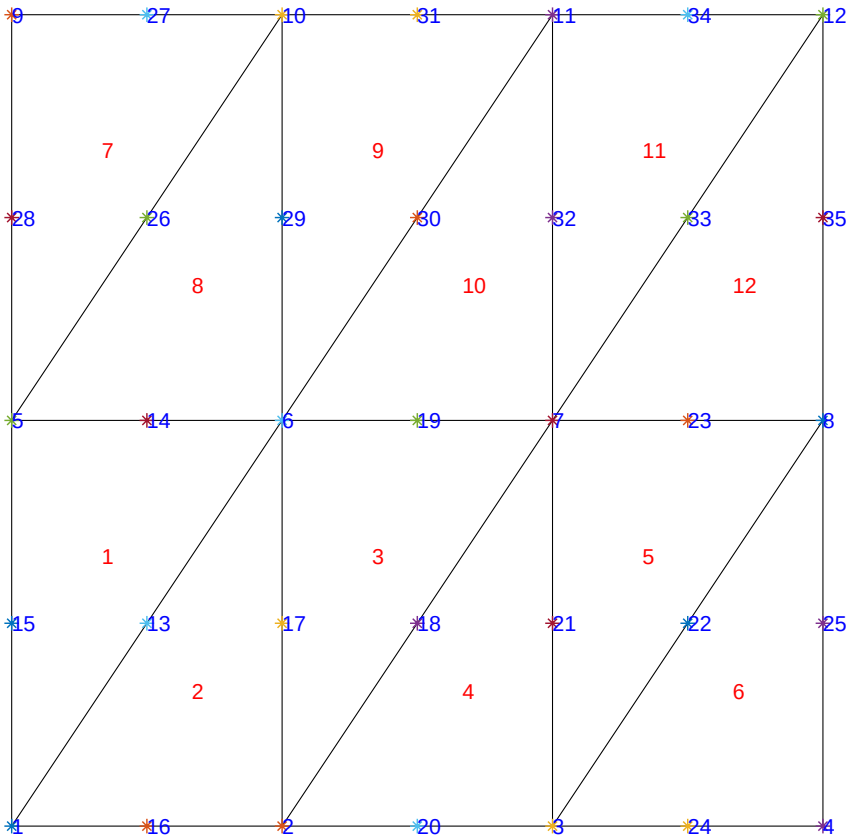
Another script for how to use `fem_generator_Lagrange_2D_tri`:

```
xmin = 0; xmax = 1; ymin = 0; ymax = 1;
nx = 3; ny = 2;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny);
figure(1); clf;
FE_mesh_viewer_2D(mesh, mesh.p, 1, 1)
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 2;
fem = fem_generator_Lagrange_2D_tri(mesh, degree);
figure(2); clf;
FE_mesh_viewer_2D(mesh, fem.p, 1, 1)
```

Mesh nodes:



FEM nodes for $V_h^2(\Omega)$:



Finite element space by mathematics: mesh $\mathcal{T}_h$, FE nodes $\mathcal{N}_{h,dof}$, FE basis functions:

$$\phi_j|_K \in V_h^p(K),\ \forall K \in \mathcal{T}_h \tag{8}$$

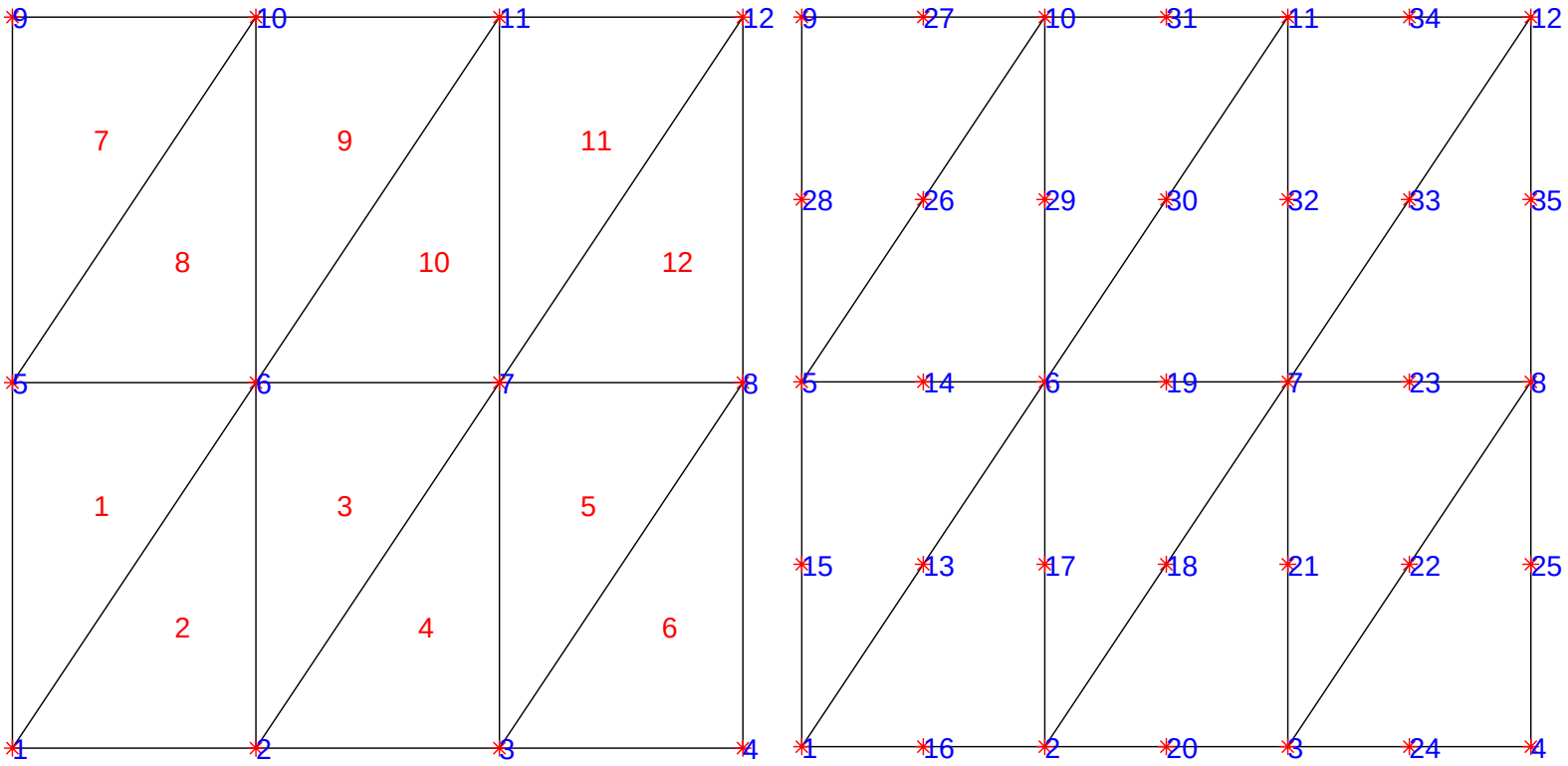
$$\phi_j(X_i) = \delta_{ij},\ \forall X_i \in \mathcal{N}_{h,dof} \qquad \text{(Lagrange property)} \tag{9}$$

and the C^0 p-th degree Lagrange type FE space:

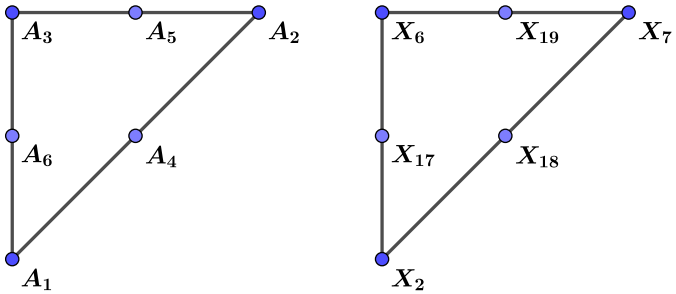
$$V_h^p(\Omega) = span\{\phi_i,\ \forall X_i \in \mathcal{N}_{h,dof}\} \tag{10}$$

Finite element space in computations:

```
mesh: by PDEtool or mesh_generator_2D_tri
shape functions on each element: by shape_fun_2D_Lagrange_tri
fem: fem_generator_Lagrange_2D_tri
```



On element K_3:



local shape functions $\Leftrightarrow$ global basis functions

$$\phi_{1,2,K_3}(X) \Leftrightarrow \phi_2(X)$$
$$\phi_{2,2,K_3}(X) \Leftrightarrow \phi_7(X)$$
$$\phi_{3,2,K_3}(X) \Leftrightarrow \phi_6(X)$$
$$\phi_{4,2,K_3}(X) \Leftrightarrow \phi_{18}(X)$$
$$\phi_{5,2,K_3}(X) \Leftrightarrow \phi_{19}(X)$$
$$\phi_{6,2,K_3}(X) \Leftrightarrow \phi_{17}(X)$$

```
fem.t(:, 3) = [2, 7, 6, 18, 19, 17]'
```

3.3, Quadrature formulas for triangular elements

Computations with finite element space $V_h^p(\Omega)$ defined on a triangular mesh $\mathcal{T}_h$ involve integrations on triangular elements. A usual numerical quadrature formula for computing the integral of $f(X) = f(x, y)$ on an element $K \in \mathcal{T}_h$ has the following form:

$$\int_K f(X)dX \approx Q(K, f) \triangleq \sum_{j=1}^{L} w_{j,K} f(X_{j,K}) \tag{13}$$

in which $w_{j,K}, 1 \leq j \leq L$ are called the quadrature weights and $X_{j,K}, 1 \leq j \leq L$ are called the quadrature nodes of the chosen quadrature formula $Q(K, f)$.

One way to describe the nodes is by their so called triangular coordinates. This is a redundant coordinate system for describing points in a triangle.

Consider a triangle $K = \triangle A_1 A_2 A_3$ whose three vertices are:

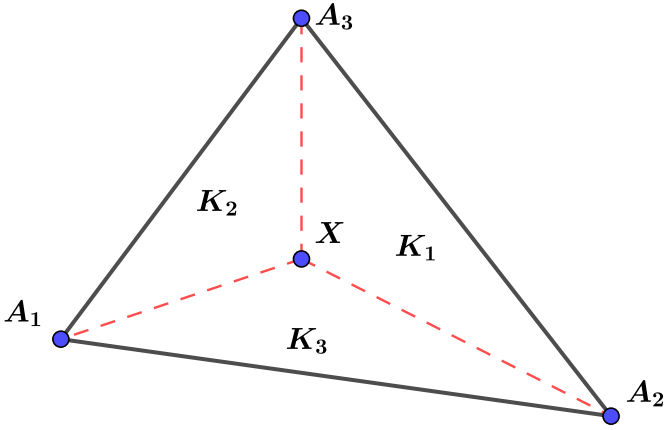
$$A_1 = (x_1, y_1), \ A_2 = (x_2, y_2), \ A_3 = (x_3, y_3)$$

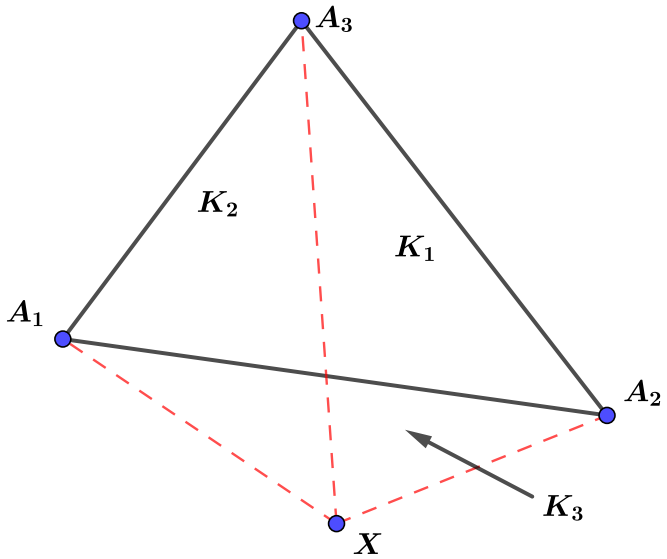
ordered counter-clock wise. The area of this triangle is

$$|K| = \frac{1}{2} det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \tag{14}$$

Given an arbitrary point $X = (x, y)$, we can form three triangles by connecting X to the three vertices

$$K_1(X) = \Delta X A_2 A_3,\ K_2(X) = \Delta A_1 X A_3,\ K_3(X) = \Delta A_1 A_2 X$$



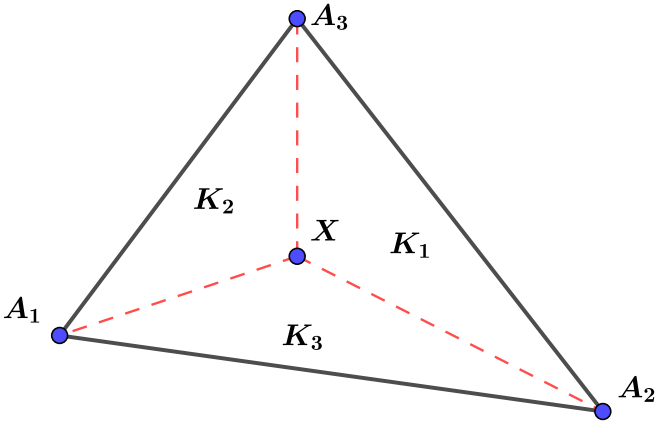


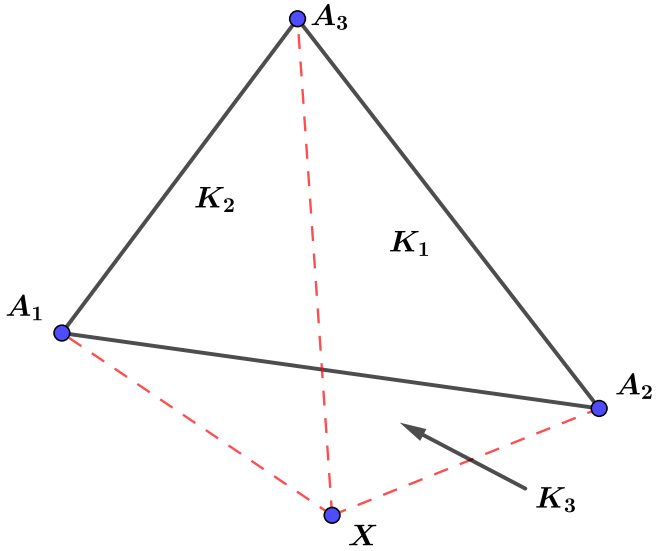
The areas of these triangles are

$$|K_1(X)| = \frac{1}{2} det \begin{bmatrix} x & y & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix},\ |K_2(X)| = \frac{1}{2} det \begin{bmatrix} x_1 & y_1 & 1 \\ x & y & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

$$|K_3(X)| = \frac{1}{2} det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{bmatrix}$$

all positive if $X \in K$





Then, we let

$$\psi_{i,K}(X) = \frac{|K_i(X)|}{|K|}, \quad i = 1, 2, 3, \tag{15}$$

and call $(\psi_{1,K}(X), \psi_{2,K}(X), \psi_{3,K}(X))$ the triangular coordinates or the area coordinates of the point $X = (x, y)$.

The above discussions provide the following transformations:

A point described by 2 axes		A point described by 3 vertices
Cartesian Coordinates: $X = (x, y)$	$\longleftrightarrow$	Tri. Coordinates: $(\psi_{1,K}(X), \psi_{2,K}(X), \psi_{3,K}(X))$

Some properties of triangular coordinates:

- $\psi_{1,K}(X) \geq 0$ for all $X \in K$.
- $\psi_{1,K}(X) + \psi_{2,K}(X) + \psi_{3,K}(X) = 1$.
- Triangular coordinates for the vertices of K:

$$A_1 = (1, 0, 0),\ A_2 = (0, 1, 0),\ A_3 = (0, 0, 1)$$

- Triangular coordinates for the middle points of the three edges of K:

$$A_4 = (1/2, 1/2, 0),\ A_5 = (0, 1/2, 1/2),\ A_6 = (1/2, 0, 1/2)$$

  A_4 is on the A_1A_2 edge, A_5 is on the A_2A_3 edge, and A_6 is on the A_3A_1 edge.
- Triangular coordinates for the center of K:

$$G_K = (1/3, 1/3, 1/3)$$

- In fact,

$$\psi_{i,K}(X) = \phi_{i,1,K}(X), \quad i = 1, 2, 3 \quad ???$$

  where $\phi_{i,1,K}(X), i = 1, 2, 3$ are three linear finite element shape functions for $V_h^1(K)$.

Transformations between the Cartesian and triangular coordinates:

The procedure above describes the following transformation:

$$X = (x, y) \longrightarrow (\psi_{1,K}(X), \psi_{2,K}(X), \psi_{3,K}(X))$$

which mean, given the x-y or the Cartesian coordinates of a point X, we can use this procedure to compute the triangular coordinates for this point.

On the other hand, given the triangular coordinates (η_1, η_2, η_3) of a point X, its Cartesian or x-y coordinates can be computed as follows:

$$\begin{aligned} X &= \begin{bmatrix} x \\ y \end{bmatrix} = \eta_1 A_1 + \eta_2 A_2 + \eta_3 A_3 && (16) \\ \text{or } x &= x(\eta_1, \eta_2, \eta_3) = x_1\eta_1 + x_2\eta_2 + x_3\eta_3 && (17) \\ y &= y(\eta_1, \eta_2, \eta_3) = y_1\eta_1 + y_2\eta_2 + y_3\eta_3 && (18) \end{aligned}$$

These formula can be derived as follows. We assume there is a matrix L such that

$$\begin{bmatrix} 1 \\ x \\ y \end{bmatrix} = L^{-1} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix} \quad \text{or} \quad L \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix} \qquad \text{then} \quad L^{-1} = ?$$

To find L^{-1}, we note that

$$A_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = L \begin{bmatrix} 1 \\ x_1 \\ y_1 \end{bmatrix}, \; A_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = L \begin{bmatrix} 1 \\ x_2 \\ y_2 \end{bmatrix}, \; A_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = L \begin{bmatrix} 1 \\ x_3 \\ y_3 \end{bmatrix}$$

Hence $$I = L \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}, \text{ and } L^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$$

Therefor, we have

$$\begin{bmatrix} 1 \\ x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix}$$

which provides the relationship: $X = (x, y) \longleftrightarrow (\eta_1, \eta_2, \eta_3)$.

We can implement these relationships as follows:

```
function X = coord_Tri2X(elem, Tri)
L = [1 1 1; elem];
X = L*Tri; X = X(2:end);
```

```
function Tri = coord_X2Tri(elem, X)
L = [1 1 1; elem]; rhs = [1; X];
Tri = L\rhs;
```

Example: Consider the triangle $K = \triangle A_1A_2A_3$ such that

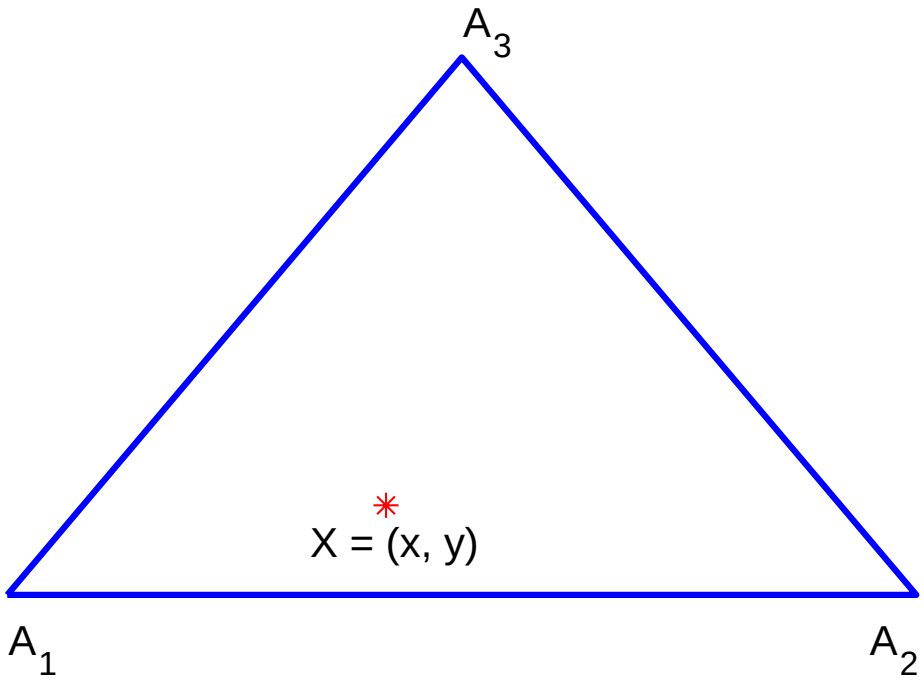
$$A_1 = (1, 1), \quad A_2 = (3, 1), \quad A_3 = (2, 2)$$

and a point X whose triangular coordinates are

$$(\eta_1, \eta_2, \eta_3) = (1/2, 1/3, 1/6)$$

Then, the Cartesian coordinates for X is

$$\begin{bmatrix} x \\ y \end{bmatrix} = \eta_1 A_1 + \eta_2 A_2 + \eta_3 A_3 = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} 3 \\ 1 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 11/6 \\ 7/6 \end{bmatrix}$$




```
elem = [1, 3, 2; 1, 1, 2];
eta = [1/2; 1/3; 1/6];
X = coord_Tri2X(elem, eta)
Tri = coord_X2Tri(elem, X)
```

Some quadrature formulas on a triangular element K:

A 1-point quadrature formula whose DOP is 1:

$$L = 1, \quad w_{1,K} = |K|, \; X_{1,K} = (1/3, 1/3, 1/3) = G_K,$$

$$\int_K f(X)dX \approx Q_1(K, f) = \sum_{j=1}^{L} w_{j,K} f(X_{j,K}) = |K| \, f(X_{1,K}) = |K| \, f(x_{1,K}, y_{1,K})$$

A 3-point quadrature formula whose DOP is 2:

$$L = 3, \quad w_{j,K} = \frac{|K|}{3}, \quad X_{j,K} = A_{j+3}, \quad j = 1, 2, 3,$$

$$\int_K f(X)dX \approx Q_3(K, f) = \sum_{j=1}^{L} w_{j,K} f(X_{j,K}) = \frac{|K|}{3} \sum_{j=1}^{3} f(A_{j+3})$$

A 7-point quadrature formula whose DOP is 5:

Weights:

$$c_1 = \frac{155 - \sqrt{15}}{1200},\ c_2 = \frac{155 + \sqrt{15}}{1200},\ c_3 = \frac{9}{40}$$
$$w_{j,K} = |K|\, c_1,\ w_{j+3,K} = |K|\, c_2,\quad j = 1, 2, 3,\quad w_{7,K} = |K|\, c_3$$

Nodes: $X_{7,K} = (1/3, 1/3, 1/3) = G_K$, and

$$s_1 = \frac{9 + 2\sqrt{15}}{21},\ t_1 = \frac{6 - \sqrt{15}}{21},\ s_2 = \frac{9 - 2\sqrt{15}}{21},\ t_2 = \frac{6 + \sqrt{15}}{21}$$
$$X_{1,K} = (x(s_1, t_1, t_1), y(s_1, t_1, t_1)),\ X_{2,K} = (x(t_1, s_1, t_1), y(t_1, s_1, t_1))$$
$$X_{3,K} = (x(t_1, t_1, s_1), y(t_1, t_1, s_1)),\ X_{4,K} = (x(s_2, t_2, t_2), y(s_2, t_2, t_2))$$
$$X_{5,K} = (x(t_2, s_2, t_2), y(t_2, s_2, t_2)),\ X_{6,K} = (x(t_2, t_2, s_2), y(t_2, t_2, s_2))$$
$$\int_K f(X)dX \approx Q_7(K, f) = \sum_{j=1}^{7} w_{j,K} f(X_{j,K})$$

Here, $x(\cdot,\cdot,\cdot)$ and $y(\cdot,\cdot,\cdot)$ are the x-y coordinates computed from the triangular coordinates according to (17) and (18).

We can develop Matlab functions for generating quadrature nodes and weights in a triangular element K as follows:

```
function qnodes = quadrature_2D_tri_nodes(elem, n_qp)
```

and

```
function qweights = quadrature_2D_tri_weights(elem, n_qp)
```

`elem` provides the vertices of element K.
`n_qp` is the number of quadrature nodes or weights to be generated.
`qnodes, qweights` are arrays containing quadrature nodes or weights. `qnodes` is a `2 x n_gp` array and `qweights` is an `1 x n_gp` array.

We implement them separately because we need to generate weights less frequently than nodes in finite element computations.

Remark: There are many quadrature formulas for integrations on a triangle, see for examples on the following web pages:

- `http://www.cs.rpi.edu/~flaherje/pdf/fea6.pdf`
- `https://people.sc.fsu.edu/~jburkardt/datasets/quadrature_rules_tri/quadrature_rules_tri.html`

```
function qnodes = quadrature_2D_tri_nodes(elem, n_qp)

qnodes = zeros(2,n_qp);

if n_qp == 1
    qnodes = (1/3)*(elem(:,1) + elem(:,2) + elem(:,3));
elseif n_qp == 3
    A4 = (1/2)*(elem(:,1) + elem(:,2));
    A5 = (1/2)*(elem(:,2) + elem(:,3));
    A6 = (1/2)*(elem(:,3) + elem(:,1));
    qnodes = [A4, A5, A6];
else
    fprintf('n_qp = %d\n', n_qp)
    disp('this program has not been implemented for this n_qp')
    disp('use Ctrl+c keys to terminate')
    pause
end

return;
```

Example: The following script generates nodes and weights on the reference triangle:

```
elem = [0, 1, 0;0, 0, 1]; n_qp = 7;
qnodes = quadrature_2D_tri_nodes(elem, n_qp)
qweights = quadrature_2D_tri_weights(elem, n_qp)
```

which produces

```
qnodes' =
     1.012865073234563e-01     1.012865073234563e-01
     7.974269853530872e-01     1.012865073234563e-01
     1.012865073234563e-01     7.974269853530872e-01
     4.701420641051151e-01     4.701420641051151e-01
     5.971587178976981e-02     4.701420641051151e-01
     4.701420641051151e-01     5.971587178976981e-02
     3.333333333333333e-01     3.333333333333333e-01
qweights' =
     6.296959027241358e-02
     6.296959027241358e-02
     6.296959027241358e-02
     6.619707639425308e-02
     6.619707639425308e-02
     6.619707639425308e-02
     1.125000000000000e-01
```

Example: We consider computing an integral on a triangular element:

$$\int_{\hat{K}} (\sin(x) + \cos(y)) dX \approx Q_7(K, f) = \sum_{j=1}^{7} w_{j,K} f(X_{j,K})$$

where $\hat{K}$ is the reference element.

```
f = @(x, y) sin(x) + cos(y);
elem = [0 1 0; 0 0 1]; n_qp = 7;
qnodes = quadrature_2D_tri_nodes(elem, n_qp);
qweights = quadrature_2D_tri_weights(elem, n_qp);
f_val = f(qnodes(1,:), qnodes(2,:));
Int = sum(qweights.*f_val);
fprintf('Int = %.16f \n', Int)
```

which produces

$$0.6182268597882430 \approx \int_{\hat{K}} (\sin(x) + \cos(y)) dX = 0.6182267093239637759 5$$

A multi-index notation for partial derivatives: for $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_d) \in \mathbb{N}_0^d$ with non-negative integers numbers $\alpha_i, i = 1, 2, \cdots, d$, we will use the following multi-index notations:

$$\begin{aligned} |\alpha| &= \alpha_1 + \alpha_2 + \cdots + \alpha_d, \quad \alpha! = \alpha_1! \alpha_2! \cdots \alpha_d! \\ X^\alpha &= x_1^{\alpha_1} \cdots x_d^{\alpha_d}, \ \forall X = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d, \quad D^\alpha = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} \end{aligned}$$

Sobolev space: For a domain $T \subset \mathbb{R}^d$, integers $m \geq 0, p \geq 1$, we let

$$W^{m,p}(T) = \{u \mid D^\alpha u \in L^p(T) \ \ \forall \alpha \text{ such that } |\alpha| \leq m\}$$

where $D^\alpha u$ is a weak partial derivative (Section 2.3.3, BK1) and

$$L^p(T) = \left\{ u : T \to \mathbb{R} \mid \int_T |u|^p \, dX < \infty \right\}$$

Read Section 2.3.1 in BK1 for more details about the $L^p(T)$ space.

Theorem 1.3

If $Q(K, f)$ is a quadrature formula of degree of precision p with $p + 1 \geq d$ on an element $K \subset \mathbb{R}^d$, then

$$\left| \int_K f(X) dX - Q(K, f) \right| \leq C h_K^{p+1} |f|_{p+1,1,K} \quad \forall f \in W^{p+1,1}(K)$$

where h_K is the diameter of K.

Assume that Ω is a polygonal domain and $\mathcal{T}_h$ is a mesh of Ω, then

$$\int_\Omega f(X)dX = \sum_{K\in\mathcal{T}_h}\int_K f(X)dX \approx \sum_{K\in\mathcal{T}_h} Q(K,f)$$

Of course, if Ω is not a polygon, then we can still expect

$$\int_\Omega f(X)dX \approx \sum_{K\in\mathcal{T}_h}\int_K f(X)dX \approx \sum_{K\in\mathcal{T}_h} Q(K,f)$$

In either case, an approximation to $\int_\Omega f(X)dX$ can be produced by quadratures over all the elements in $\mathcal{T}_h$.

Theorem 1.4

Assume that $\Omega \subset \mathbb{R}^d$ is a polygon and $\mathcal{T}_h$ is a mesh of Ω such that $\overline{\Omega} = \cup_{K\in\mathcal{T}_h} K$. Also, assume $Q(K,f)$ is a quadrature formula of degree of precision p with $p+1 \geq d$. Then

$$\left|\int_\Omega f(X)dX - \sum_{K\in\mathcal{T}_h} Q(K,f)\right| \leq Ch^{p+1}\,|f|_{p+1,1,\Omega} \quad \forall f \in W^{p+1,1}(\Omega)$$

where $h = \max_{K\in\mathcal{T}_h} h_K$ and h_K is the diameter of $K \in \mathcal{T}_h$.

Example: We consider computing an integral on a mesh:

$$\int_{\Omega} (\sin(x) + \cos(y)) dX$$

where $\Omega = (0, 1) \times (0, 1)$ is the unit square.

```
mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
f = @(x, y) sin(x) + cos(y);
Int = 0; n_qp = 7;
for k = 1:length(mesh.t) % integrate on each element
    elem = mesh.p(:, mesh.t(:, k));
    qnodes = quadrature_2D_tri_nodes(elem, n_qp);
    qweights = quadrature_2D_tri_weights(elem, n_qp);
    f_val = f(qnodes(1,:), qnodes(2,:));
    Int = Int + sum(qweights.*f_val);
end
fprintf('Int = %.16f \n', Int)
```

which produces

$$\begin{aligned} 1.301168678939762 &\approx \int_{\Omega} (\sin(x) + \cos(y)) dX \\ &= 1.3011686789397\,6789251565714\,19 \end{aligned}$$

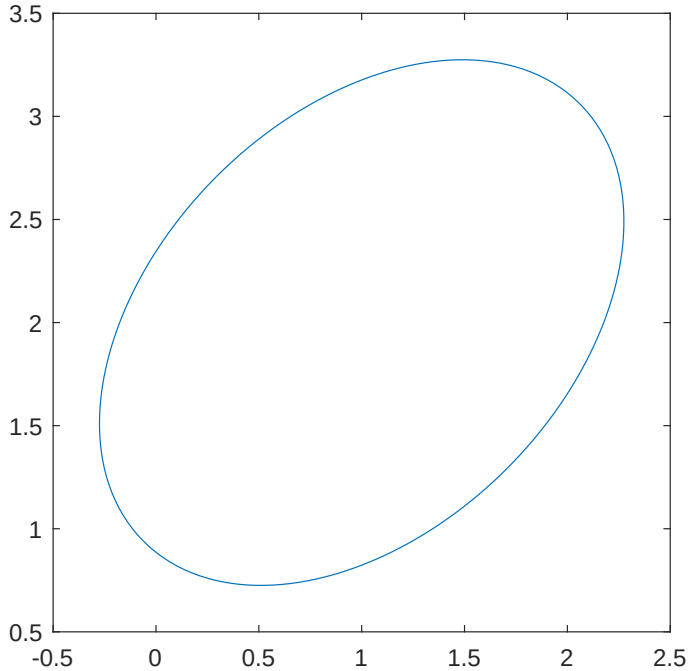
Example: Compute

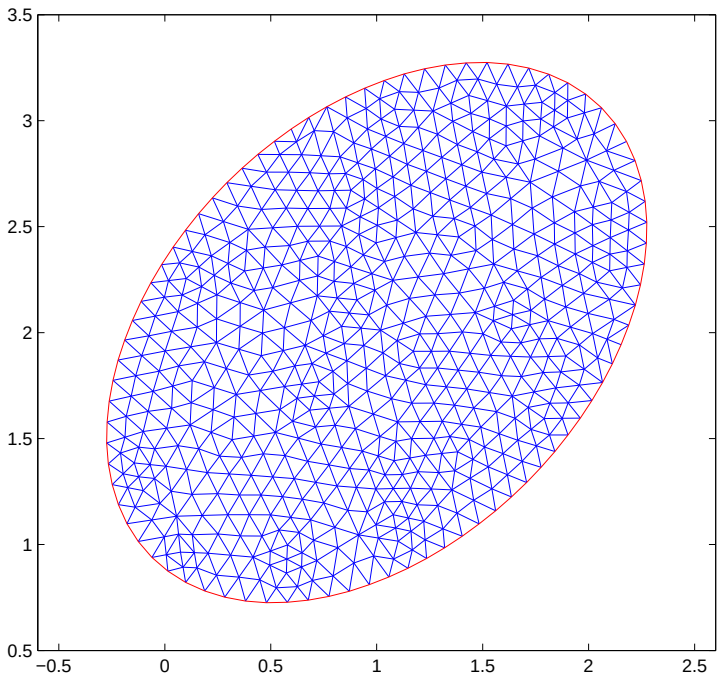
$$\int_{\Omega} (\sin(x) + \cos(y)) dX$$

where Ω is the elliptic disk formed by rotating the following elliptic curve $\pi/4$ degree:

$$\frac{x_1^2}{1.5^2} + \frac{x_2^2}{1^2} = 1$$

Plots for the integration domain Ω and a mesh:





```
C1 = [4, 1, 2, 1.5, 1, pi/4]';
gd = decsg(C1);
[p, e, t] = initmesh(gd); n_refine = 3;
for i = 1:n_refine
    [p, e, t] = refinemesh(gd, p, e, t);
end
mesh.p = p; mesh.t = t(1:3, :); % form a basic mesh
f = @(x, y) sin(x) + cos(y);
Int = 0; n_qp = 3;
for k = 1:length(mesh.t) % integrate on each element
    elem = mesh.p(:, mesh.t(:, k));
    qnodes = quadrature_2D_tri_nodes(elem, n_qp);
    qweights = quadrature_2D_tri_weights(elem, n_qp);
    f_val = f(qnodes(1,:), qnodes(2,:));
    Int = Int + sum(qweights.*f_val);
end
fprintf('Int = %.16f \n', Int)
```

which produces `Int = 1.6237390108804297`.

3.4, Matrix and vector assemblers for 2D finite element spaces:

Our experiences in solving 1D BVPs with finite element methods tells us that matrices and vectors formed by the basis functions of a finite element space are key components of finite element methods.

Also, the element-by-element assembling procedures developed for matrices and vectors of 1D finite element spaces extend to 2D finite element spaces.

Consider a domain $\Omega \subset \mathbb{R}^2$ and its mesh $\mathcal{T}_h$. On this mesh, we assume we have two C^0 finite element spaces:

$$V_h^{p_1}(\Omega) = span\left\{\phi_{i,1},\ \forall X_i \in \mathcal{N}_{h,dof}^1\right\}, \quad V_h^{p_2}(\Omega) \quad = span\left\{\phi_{i,2},\ \forall X_i \in \mathcal{N}_{h,dof}^2\right\}$$

and recall the local p-th degree finite element space on each element $K \in \mathcal{T}_h$:

$$V_h^{p_k}(K) = span\left\{\phi_{i,p_k,K},\ i = 1, 2, \cdots, \frac{(p_k+1)(p_k+2)}{2}\right\}, \quad k = 1, 2$$

We want to construct the following vector and matrix:

$$\begin{aligned} \vec{b} &= \left(\int_\Omega \left(D^\alpha \phi_{i,1}(X)\right) f(X) dX\right)_{i=1}^{|\mathcal{N}_{h,g}^1|}, \quad 0 \le |\alpha| \le 1 \\ M &= \left(\int_\Omega a(X)\left(D^{\alpha_1}\phi_{i,1}(X)\right)\left(D^{\alpha_2}\phi_{j,2}(X)\right) dx\right)_{i,\ j=1}^{|\mathcal{N}_{h,g}^1|,\ |\mathcal{N}_{h,g}^2|} \\ & \quad 0 \le |\alpha_1| \le 1, \quad 0 \le |\alpha_2| \le 1 \end{aligned}$$

where we have use the multi-index notations such that $\alpha = (d_x, d_y)$ and

$$D^\alpha \phi_{i,1}(X) = \partial_x^{d_x} \partial_y^{d_y} \phi_{i,j}(X), \quad j = 1, 2$$

Since the computations for the vector involve one FE space, we let $p_1 = p$ so that

$$V_h^{p_1}(K) = V_h^p(K) = span\left\{\phi_{i,p,K},\ i = 1, 2, \cdots, \frac{(p+1)(p+2)}{2}\right\}$$

The local vector assembler: On an element $K \in \mathcal{T}_h$, we consider the following vector for a finite element space $V_h^p(\Omega)$:

$$\begin{aligned} \vec{b}_l &= \left(\int_K \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{i,p,K}(X)\right) f(X)dX\right)_{i=1}^{ldof}, \quad 0 \le d_x + d_y \le 1 \\ ldof &= (p+1)(p+2)/2 \end{aligned}$$

We can use the shape function `shape_fun_2D_Lagrange_tri` to implement this as follows:

```
function b_l = FE_vec_2D_Lagrange_tri_loc(f_name, elem, ...
                    degree, d_x, d_y, n_qp)
qnodes = quadrature_2D_tri_nodes(elem, n_qp);
qweights = quadrature_2D_tri_weights(elem, n_qp);
ldof = (degree + 1)*(degree + 2)/2; b_l = zeros(ldof, 1);
f_val = feval(f_name, qnodes(1,:), qnodes(2,:));
for i=1:ldof % integration for each shape function
    ibas_val = shape_fun_2D_Lagrange_tri(qnodes(1,:), qnodes(2,:), ...
                    elem, degree, i, d_x, d_y);
    b_l(i) = sum(qweights.*ibas_val.*f_val);
end
```

A comparison of the 1D and 2D local vector assemblers:

```
function b_l = FE_vec_1D_Lagrange_loc(f_name, elem, degree, ...
    d_index, n_gp)
nodes = Gauss_1D_nodes_local(elem, n_gp);
weights = Gauss_1D_weights_local(elem, n_gp);
f = feval(f_name, nodes);
b_l = zeros(degree + 1, 1);
for i = 1:(degree+1)
    L_i = shape_fun_1D_Lagrange(nodes, elem, degree, i, d_index);
    b_l(i) = sum(weights.*L_i.*f);
end
```

```
function b_l = FE_vec_2D_Lagrange_tri_loc(f_name, elem, ...
                    degree, d_x, d_y, n_qp)

qnodes = quadrature_2D_tri_nodes(elem, n_qp);
qweights = quadrature_2D_tri_weights(elem, n_qp);
ldof = (degree + 1)*(degree + 2)/2; b_l = zeros(ldof, 1);
f_val = feval(f_name, qnodes(1,:), qnodes(2,:));
for i=1:ldof
    ibas_val = shape_fun_2D_Lagrange_tri(qnodes(1,:), qnodes(2,:), ...
                   elem, degree, i, d_x, d_y);
    b_l(i) = sum(qweights.*ibas_val.*f_val);
end
```

Example: Use the 7 point quadrature formula to compute the local load vector for the finite element space $V_h^1(K)$ for $f(x, y) = \sin(x) + \cos(y)$ on an element $K = \triangle A_1 A_2 A_3$ whose vertices are

$$A_1 = (0, 1),\ A_2 = (0.1, 1.1),\ A_3 = (0.05, 1.2)$$

By the following Matlab script

```
n_qp = 7; elem = [0 0.1 0.05;1 1.1 1.2];
f_name = @(x, y) sin(x) + cos(y);
degree = 1;
d_x = 0; d_y = 0;
b_l = FE_vec_2D_Lagrange_tri_loc(f_name, elem, ...
            degree, d_x, d_y, n_qp)
```

we obtain the local load vetor

```
b_l = 1.28223099773309e-03
      1.28954587648429e-03
      1.20212568911387e-03
```

Example: Use the 7 point quadrature formula to compute the local load vector for the finite element space $V_h^2(K)$ for $f(x, y) = sin(x) + cos(y)$ on an element $K = \triangle A_1 A_2 A_3$ whose vertices are

$$A_1 = (0, 1),\ A_2 = (0.1, 1.1),\ A_3 = (0.05, 1.2)$$

By the following Matlab script

```
n_qp = 7; elem = [0 0.1 0.05;1 1.1 1.2];
f_name = @(x, y) sin(x) + cos(y);
degree = 2;
d_x = 0; d_y = 0;
b_l = FE_vec_2D_Lagrange_tri_loc(f_name, elem, ...
            degree, d_x, d_y, n_qp)
```

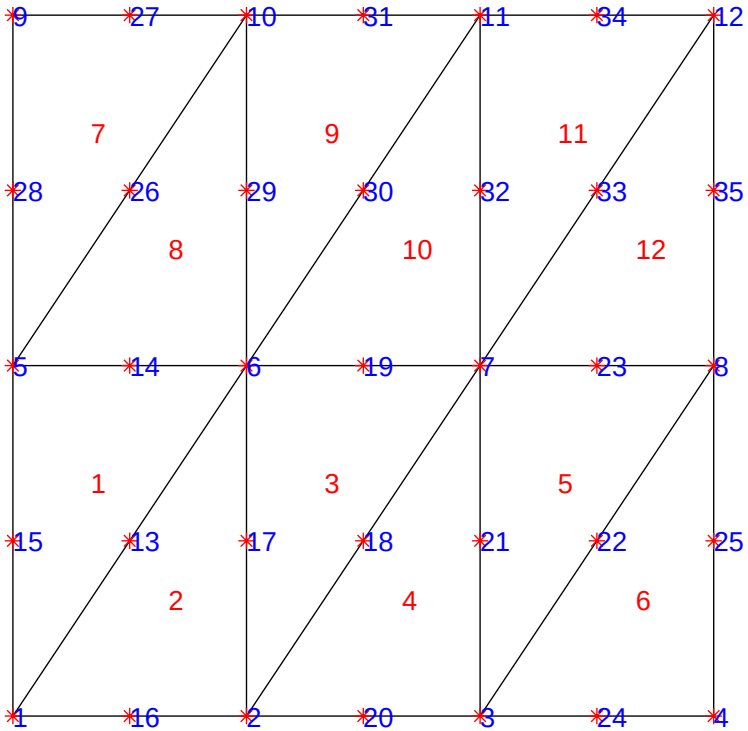
we obtain the local load vetor

```
b_l = 1.428265904150992e-05
      1.890136847845604e-05
     -3.376638876525498e-05
      1.302700768860992e-03
      1.238588247150676e-03
      1.233195908602608e-03
```

The global vector assembler: On Ω, we consider the following vector:

$$\vec{b} = \left(\int_\Omega \left(\partial_x^{d_x}\partial_y^{d_y}\phi_i(X)\right) f(X)dX\right)_{i=1}^{|\mathcal{N}_{h,g}|} = \left(\sum_{K\in\mathcal{T}_h}\int_K \left(\partial_x^{d_x}\partial_y^{d_y}\phi_i(X)\right) f(X)dX\right)_{i=1}^{|\mathcal{N}_{h,g}|}$$

$dim(\vec{b}) = 35$ for $V_h^2(\Omega)$ on the following mesh:



and $dim(\vec{b}_l) = 6$ on each element K which provides all needed computations with the global basis functions on K.

On element K_3:

$$\vec{b}_l = \begin{bmatrix} \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{1,2,K_3}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{2,2,K_3}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{3,2,K_3}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{4,2,K_3}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{5,2,K_3}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{6,2,K_3}(X)\right) f(X)dX \end{bmatrix}$$

$$= \begin{bmatrix} \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_2(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_7(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_6(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{18}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{19}(X)\right) f(X)dX \\ \int_{K_3}\left(\partial_x^{d_x}\partial_y^{d_y}\phi_{17}(X)\right) f(X)dX \end{bmatrix}$$

$$\vec{b}_l = \begin{bmatrix} \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{1,2,K_3}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{2,2,K_3}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{3,2,K_3}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{4,2,K_3}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{5,2,K_3}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{6,2,K_3}(X)\right) f(X)dX \end{bmatrix} = \begin{bmatrix} \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{2}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{7}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{6}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{18}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{19}(X)\right) f(X)dX \\ \int_{K_3} \left(\partial_x^{d_x}\partial_y^{d_y}\phi_{17}(X)\right) f(X)dX \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_9 \\ b_{10} \\ b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \\ \vdots \\ b_{35} \end{bmatrix} = \vec{b}$$

We consider the following vector:

$$\vec{b} = \left(\int_\Omega (\partial_x^{d_x} \partial_y^{d_y} \phi_i(X)) f(X) dX \right)_{i=1}^{|\mathcal{N}_{h,g}|} = \left(\sum_{K \in \mathcal{T}_h} \int_K (\partial_x^{d_x} \partial_y^{d_y} \phi_i(X)) f(X) dX \right)_{i=1}^{|\mathcal{N}_{h,g}|}$$

Discussions above means that we can use

```
function b_l = FE_vec_2D_Lagrange_tri_loc(f_name, elem, ...
                    degree, d_x, d_y, n_qp)
```

to generate this vector by a Matlab function with the following interface:

```
function b = FE_vec_2D_Lagrange_tri(f_name, mesh, fem, ...
                        d_x, d_y, n_qp)
```

and this Matlab function can be easily developed by modifying its corresponding 1D code:

```
function b = FE_vec_1D_Lagrange(f_name, mesh, fem, d_index, n_gp)
degree = fem.degree;
b = zeros(size(fem.p, 2), 1);
for k = 1:size(mesh.t,2)
    elem = mesh.p(mesh.t(:,k));
    b_l = FE_vec_1D_Lagrange_loc(f_name, elem, degree, d_index, n_gp);
    b(fem.t(:,k)) = b(fem.t(:,k)) + b_l;
end
```

Example: The following script demonstrate how to generated a load vector on the domain $\Omega = (0,1) \times (0,1)$ for the function $f(x,y) = \sin(x) + \cos(y)$ and $V_h^1(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 10, 10);
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 1;
fem = fem_generator_Lagrange_2D_tri(mesh, degree);
f_name = @(x, y) sin(x) + cos(y); d_x = 0; d_y = 0; n_qp = 7;
b = FE_vec_2D_Lagrange_tri(f_name, mesh, fem, ...
                           d_x, d_y, n_qp);
b(70:75)
```

This produces a vector b whose dimension is dim(b) = size(fem.p) = 121 and some entries of b are

```
b(70:75) = 1.119922086345015e-02
           1.213741999624004e-02
           1.303674214920077e-02
           1.388820159265630e-02
           1.468329082530273e-02
           1.541406557834900e-02
```

Example: The following script demonstrate how to generated a load vector on the domain $\Omega = (0,1) \times (0,1)$ for the function $f(x,y) = \sin(x) + \cos(y)$ and $V_h^2(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 10, 10);
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 2; % the same mesh as before but quadratic FE
fem = fem_generator_Lagrange_2D_tri(mesh, degree);
f_name = @(x, y) sin(x) + cos(y); d_x = 0; d_y = 0; n_qp = 7;
b = FE_vec_2D_Lagrange_tri(f_name, mesh, fem, ...
                           d_x, d_y, n_qp);
b(70:75)
```

Even though the mesh is the same, this script produces a larger vector `b` whose dimension is `dim(b) = size(fem.p) = 441` and some entries of `b` are

```
b(70:75) = 3.111900243241774e-06
           3.372595352760385e-06
           3.622487810519653e-06
           3.859080773677994e-06
           4.080010283568920e-06
           4.283068885551044e-06
```

Local matrix assembler on each element: On each element K in a mesh $\mathcal{T}_h$, consider the following matrix

$$M_l = \left(\int_K a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,p_1,K}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,p_2,K}(X)\right) dX \right)_{i,j=1}^{ldof_1, ldof_2}$$

$$ldof_1 = (p_1+1)(p_1+2)/2, \quad ldof_2 = (p_2+1)(p_2+2)/2$$

$$0 \le d_{x,1} + d_{y,1} \le 1, \quad 0 \le d_{x,2} + d_{y,2} \le 1$$

where $a(X)$ is a given coefficient function, $\phi_{i,p_1,K}(X), 1 \le i \le ldof_1$ and $\phi_{i,p_2,K}(X), 1 \le i \le ldof_2$ are shape functions for the local finite element spaces $V_h^{p_1}(K)$ and $V_h^{p_2}(K)$ defined on element K, respectively.

We can develop a Matlab function for generating this local matrix on any element $K \in \mathcal{T}_h$ such that its interface is as follows:

```
function M_l = FE_matrix_2D_Lagrange_tri_loc(a_name, elem, ...
               degree1, d_x1, d_y1, ...
               degree2, d_x2, d_y2, n_qp)
```

Again, we can develop this 2D matrix assembler by modifying its corresponding 1D Matlab function:

```
function M_l = FE_matrix_1D_Lagrange_loc(a_name, elem, ...
               degree1, d_index1, ...
               degree2, d_index2, n_gp)
```

```
function M_l = FE_matrix_1D_Lagrange_loc(a_name, elem, ...
                    degree1, d_index1, ...
                    degree2, d_index2, n_gp)
weights = Gauss_1D_weights_local(elem, n_gp); nodes = Gauss_1D_nodes_local(elem, n_gp);
a_val = feval(a_name, nodes);
M_l = zeros(degree1 + 1, degree2 + 1);
for i = 1:(degree1 + 1)
    L_i = shape_fun_1D_Lagrange(nodes, elem, degree1, i, d_index1);
    for j = 1:(degree2 + 1)
        L_j = shape_fun_1D_Lagrange(nodes, elem, degree2, j, d_index2);
        f = a_val.*L_i.*L_j;
        M_l(i,j) = sum(weights.*f);
    end
end
```

```
function M_l = FE_matrix_2D_Lagrange_tri_loc(a_name, elem, ...
                degree1, d_x1, d_y1, ...
                degree2, d_x2, d_y2, n_qp)
qweights = quadrature_2D_tri_weights(elem, n_qp); qnodes = quadrature_2D_tri_nodes(elem, n_qp);
a_val = feval(a_name, qnodes(1,:), qnodes(2,:));
ldof_1 = (degree1 + 1)*(degree1 + 2)/2; ldof_2 = (degree2 + 1)*(degree2 + 2)/2;
M_l = zeros(ldof_1, ldof_2);
for i=1:ldof_1
     ibas_val = shape_fun_2D_Lagrange_tri(qnodes(1,:), qnodes(2,:), elem, ...
                    degree1, i, d_x1, d_y1);
    for j=1:ldof_2
        jbas_val = shape_fun_2D_Lagrange_tri(qnodes(1,:), qnodes(2,:), elem, ...
                    degree2, j, d_x2, d_y2);
        eint = sum(qweights.*a_val.*ibas_val.*jbas_val);
        M_l(i,j) = M_l(i,j)  + eint;
    end
end
```

Example: Let $K = \triangle A_1A_2A_3$ be the element whose vertices are

$$A_1 = (0, 1),\ A_2 = (0.1, 1.1),\ A_3 = (0.05, 1.2)$$

Use the 7 point quadrature formula to compute the following local matrices for the finite element space $V_h^1(K)$ in which $a(x, y) = \sin(x) + \cos(y)$:

$$\begin{aligned}
M_1 &= \left(\int_K a(X)\phi_{i,p_1,K}(X)\phi_{j,p_2,K}(X) \right)_{i,j=1}^{ldof_1,ldof_2} \\
M_2 &= \left(\int_K a(X)\big(\partial_x\phi_{i,p_1,K}(X)\big)\big(\partial_x\phi_{j,p_2,K}(X)\big) \right)_{i,j=1}^{ldof_1,ldof_2} \\
M_3 &= \left(\int_K a(X)\big(\partial_y\phi_{i,p_1,K}(X)\big)\big(\partial_y\phi_{j,p_2,K}(X)\big) \right)_{i,j=1}^{ldof_1,ldof_2}
\end{aligned}$$

By the following Matlab script

```
clear;
n_qp = 7; elem = [0 0.1 0.05;1 1.1 1.2];
a_name = @(x, y) sin(x) + cos(y);
degree1 = 1; degree2 = 1;
d_x1 = 0; d_y1 = 0; d_x2 = 0; d_y2 = 0;
M_1 =  FE_matrix_2D_Lagrange_tri_loc(a_name, elem, ...
        degree1, d_x1, d_y1, ...
        degree2, d_x2, d_y2, n_qp)
d_x1 = 1; d_y1 = 0; d_x2 = 1; d_y2 = 0;
M_2 =  FE_matrix_2D_Lagrange_tri_loc(a_name, elem, ...
        degree1, d_x1, d_y1, ...
        degree2, d_x2, d_y2, n_qp)
d_x1 = 0; d_y1 = 1; d_x2 = 0; d_y2 = 1;
M_3 =  FE_matrix_2D_Lagrange_tri_loc(a_name, elem, ...
        degree1, d_x1, d_y1, ...
        degree2, d_x2, d_y2, n_qp)
```

we obtain

```
M_1 = 6.482568284074098e-04   3.256751922152479e-04   3.082989771506519e-04
      3.256751922152478e-04   6.542236224813730e-04   3.096470617876691e-04
      3.082989771506519e-04   3.096470617876691e-04   5.841796501730661e-04

M_2 = 1.677290028163992e-01  -3.354580056327987e-01   1.677290028163996e-01
     -3.354580056327987e-01   6.709160112655982e-01  -3.354580056327994e-01
      1.677290028163996e-01  -3.354580056327994e-01   1.677290028163999e-01
```

and

```
M_3 = 4.193225070409987e-02   4.193225070409990e-02  -8.386450140819977e-02
      4.193225070409990e-02   4.193225070409992e-02  -8.386450140819982e-02
     -8.386450140819977e-02  -8.386450140819982e-02   1.677290028163996e-01
```

Example: We can also modify the previous Matlab script for computing the local matrices for $V_h^2(K)$. For example, with 7 quadrature nodes, we have

```
M_1 =

  Columns 1 through 3

   1.312342807242013e-04   -2.359828847317922e-05   -1.941708526324727e-05
  -2.359828847317922e-05    1.330055126632941e-04   -1.978495212451485e-05
  -1.941708526324726e-05   -1.978495212451486e-05    1.129306028551452e-04
   2.389245635303649e-06    3.860713080373707e-06   -8.814319517703954e-05
  -8.203916418291334e-05    6.912728577435536e-06   -9.812084262289441e-06
   5.713670601344885e-06   -8.149434524495325e-05   -9.539674793309092e-06

  Columns 4 through 6

   2.389245635303663e-06   -8.203916418291335e-05    5.713670601344885e-06
   3.860713080373700e-06    6.912728577435536e-06   -8.149434524495325e-05
  -8.814319517703954e-05   -9.812084262289448e-06   -9.539674793309098e-06
   7.049542057584624e-04    3.403647187199058e-04    3.392750808439856e-04
   3.403647187199058e-04    6.560950294428045e-04    3.270670188557332e-04
   3.392750808439856e-04    3.270670188557332e-04    6.521741583398063e-04
```

Global matrix assembler on Ω: Given/Choosing a mesh $\mathcal{T}_h$ of Ω and two FE spaces:

$$V_h^{p_1}(\Omega) = Span\{\phi_{i,1}(X) \mid i \in \mathcal{N}_{h,g}^1\}, \quad V_h^{p_2}(\Omega) = Span\{\phi_{j,2}(X) \mid j \in \mathcal{N}_{h,g}^2\}$$

we consider the construction of the following matrix

$$\begin{aligned} M &= \left(\int_\Omega a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,1}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,2}(X)\right) dx \right)_{i,\, j=1}^{\left|\mathcal{N}_{h,g}^1\right|,\ \left|\mathcal{N}_{h,g}^2\right|} \\ & 0 \le d_{x,1} + d_{y,1} \le 1, \quad 0 \le d_{x,2} + d_{y,2} \le 1 \end{aligned}$$

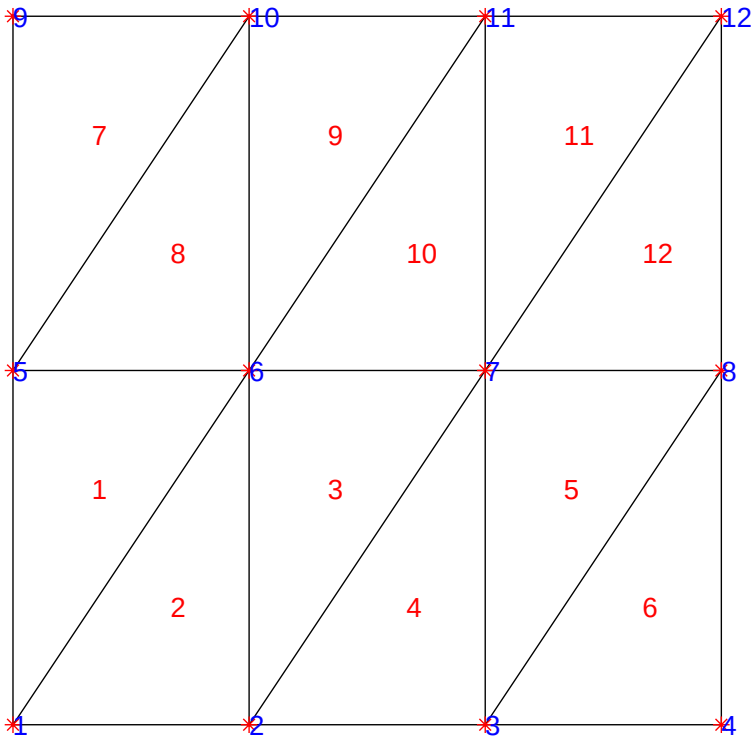
Recall that we generally do not compute with the global basis functions directly:

$$\phi_{i,1}(X),\ 1 \le i \le \left|\mathcal{N}_{h,g}^1\right|, \qquad \phi_{j,2}(X),\ 1 \le j \le \left|\mathcal{N}_{h,g}^2\right|$$

However,

$$\begin{aligned} M &= \left(\int_\Omega a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,1}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,2}(X)\right) dx \right)_{i,\, j=1}^{\left|\mathcal{N}_{h,g}^1\right|,\ \left|\mathcal{N}_{h,g}^2\right|} \\ &= \left(\sum_{K \in \mathcal{T}_h} \int_K a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,1}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,2}(X)\right) dx \right)_{i,\, j=1}^{\left|\mathcal{N}_{h,g}^1\right|,\ \left|\mathcal{N}_{h,g}^2\right|} \end{aligned}$$

which suggests that we can compute the components for the entries of this matrix through all the elements in the mesh $\mathcal{T}_h$.



On this mesh, $dim(V_h^1(\Omega)) = 12$.
Hence, $M = M_{12\times 12}$.

On element K_3, we have

`mesh.t(:, 3) = [2, 7, 6]'`

$$M_l = \left(\int_{K_3} a(X)\phi_{i,1,K}(X)\phi_{j,1,K}(X) \right)_{i,j=1}^{3,3}$$

$$= \begin{bmatrix} \int_{K_3} a(X)\phi_{1,1,K}(X)\phi_{1,1,K}(X) & \int_{K_3} a(X)\phi_{1,1,K}(X)\phi_{2,1,K}(X) & \int_{K_3} a(X)\phi_{1,1,K}(X)\phi_{3,1,K}(X) \\ \int_{K_3} a(X)\phi_{2,1,K}(X)\phi_{1,1,K}(X) & \int_{K_3} a(X)\phi_{2,1,K}(X)\phi_{2,1,K}(X) & \int_{K_3} a(X)\phi_{2,1,K}(X)\phi_{3,1,K}(X) \\ \int_{K_3} a(X)\phi_{3,1,K}(X)\phi_{1,1,K}(X) & \int_{K_3} a(X)\phi_{3,1,K}(X)\phi_{2,1,K}(X) & \int_{K_3} a(X)\phi_{3,1,K}(X)\phi_{3,1,K}(X) \end{bmatrix}$$

$$= \begin{bmatrix} \int_{K_3} a(X)\phi_2(X)\phi_2(X) & \int_{K_3} a(X)\phi_2(X)\phi_7(X) & \int_{K_3} a(X)\phi_2(X)\phi_6(X) \\ \int_{K_3} a(X)\phi_7(X)\phi_2(X) & \int_{K_3} a(X)\phi_7(X)\phi_7(X) & \int_{K_3} a(X)\phi_7(X)\phi_6(X) \\ \int_{K_3} a(X)\phi_6(X)\phi_2(X) & \int_{K_3} a(X)\phi_6(X)\phi_7(X) & \int_{K_3} a(X)\phi_6(X)\phi_6(X) \end{bmatrix}$$

We can develop a Matlab function for generating the following matrix

$$
\begin{aligned}
M &= \left(\int_\Omega a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,1}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,2}(X)\right) dx \right)_{i,\, j=1}^{|\mathcal{N}_{h,g}^1|,\ |\mathcal{N}_{h,g}^2|} \\
&= \left(\sum_{K \in \mathcal{T}_h} \int_K a(X) \left(\partial_x^{d_{x,1}} \partial_y^{d_{y,1}} \phi_{i,1}(X)\right) \left(\partial_x^{d_{x,2}} \partial_y^{d_{y,2}} \phi_{j,2}(X)\right) dx \right)_{i,\, j=1}^{|\mathcal{N}_{h,g}^1|,\ |\mathcal{N}_{h,g}^2|}
\end{aligned}
$$

such that its interface is as follows:

```
function M = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem1, d_x1, d_y1, fem2, d_x2, d_y2, n_qp)
```

Again, this Matlab function can be easily developed by modifying its 1D counterpart:

```
function M = FE_matrix_1D_Lagrange(a_name, mesh, fem1, d_index1, ...
                                                 fem2, d_index2, n_gp)
degree1 = fem1.degree; degree2 = fem2.degree;
M = sparse(size(fem1.p, 2), size(fem2.p, 2));
for k = 1:size(mesh.t,2)
    elem = mesh.p(mesh.t(:,k));
    M_l = FE_matrix_1D_Lagrange_loc(a_name, elem, degree1, d_index1, ...
                                                  degree2, d_index2, n_gp);
    M(fem1.t(:,k), fem2.t(:,k)) = M(fem1.t(:,k), fem2.t(:,k)) + M_l;
end
```

Example: The following script demonstrate how to generated a matrix on the domain $\Omega = (0,1) \times (0,1)$ for the function $a(x,y) = 2 + \sin(x) + \cos(y)$ and $V_h^1(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 1; fem = fem_generator_Lagrange_2D_tri(mesh, degree);
a_name = @(x, y) 2 + sin(x) + cos(y); d_x = 0; d_y = 0; n_qp = 7;
M_00 = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem, d_x, d_y, ...
                fem, d_x, d_y, n_qp);
```

Since `size(fem.p, 2) = 10201`, this matrix `M_00` generated for $V_h^1(\Omega)$ is quite large, but it is also a very sparse banded matrix. We can visualize its sparsity in Matlab by `spy(M_00)`.

Here are a few entries of M:

```
M_00(23:28, 23:28) =
    (1,1)         8.048810113895076e-05
    (2,1)         1.343365526418934e-05
    (1,2)         1.343365526418934e-05
    (2,2)         8.073172292831382e-05
    (3,2)         1.347421492784551e-05
    (2,3)         1.347421492784551e-05
    (3,3)         8.097477153627165e-05
    (4,3)         1.351467716943185e-05
    (3,4)         1.351467716943185e-05
    (4,4)         8.121722265816594e-05
    (5,4)         1.355503794275800e-05
    (4,5)         1.355503794275800e-05
    (5,5)         8.145905204908640e-05
    (6,5)         1.359529321178005e-05
    (5,6)         1.359529321178005e-05
    (6,6)         8.170023552629634e-05
```

These data can be used for checking your program.

Example: The following script demonstrate how to generated a matrix on the domain $\Omega = (0,1) \times (0,1)$ for the function $a(x,y) = 2 + \sin(x) + \cos(y)$ and $V_h^2(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
C = 1; B = 1;
[e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 2; fem = fem_generator_Lagrange_2D_tri(mesh, degree);
a_name = @(x, y) 2 + sin(x) + cos(y); d_x = 0; d_y = 0; n_qp = 7;
M_00 = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem, d_x, d_y, ...
                fem, d_x, d_y, n_qp);
```

Since `size(fem.p, 2) = 40401`, this matrix `M_00` generated for $V_h^2(\Omega)$ is even larger and it is also a very sparse banded matrix. We can visualize its sparsity in Matlab by `spy(M_00)`.

Here are a few entries of M

```
M_00(23:28, 23:28) =
    (1,1)         1.609577519756437e-05
    (2,1)        -8.951154288565463e-07
    (1,2)        -8.951154288565464e-07
    (2,2)         1.614450391556189e-05
    (3,2)        -8.978204757235227e-07
    (2,3)        -8.978204757235230e-07
    (3,3)         1.619311818247773e-05
    (4,3)        -9.005190741347031e-07
    (3,4)        -9.005190741347033e-07
    (4,4)         1.624161313692534e-05
    (5,4)        -9.032109542325179e-07
    (4,5)        -9.032109542325179e-07
    (5,5)         1.628998392944989e-05
    (6,5)        -9.058958468311670e-07
    (5,6)        -9.058958468311670e-07
    (6,6)         1.633822572301291e-05
```

Example: The following script demonstrate how to generated a matrix on the domain $\Omega = (0,1) \times (0,1)$ for the function $a(x,y) = 2 + \sin(x) + \cos(y)$ and $V_h^1(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
C = 1; B = 1; [e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 1; fem = fem_generator_Lagrange_2D_tri(mesh, degree);
a_name = @(x, y) 2 + sin(x) + cos(y); d_x = 1; d_y = 0; n_qp = 7;
M_xx = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem, d_x, d_y, ...
                fem, d_x, d_y, n_qp);

M_xx(23:27, 23:27) =
   (1,1)        3.219844123984656e+00
   (2,1)       -1.612360876000487e+00
   (1,2)       -1.612360876000487e+00
   (2,2)        3.229588143606682e+00
   (3,2)       -1.617227267606195e+00
   (2,3)       -1.617227267606195e+00
   (3,3)        3.239309203772352e+00
   (4,3)       -1.622081936166158e+00
   (3,4)       -1.622081936166158e+00
   (4,4)        3.249006332383736e+00
   (5,4)       -1.626924396217579e+00
   (4,5)       -1.626924396217579e+00
   (5,5)        3.258678559736058e+00
```

Example: The following script demonstrate how to generated a matrix on the domain $\Omega = (0,1) \times (0,1)$ for the function $a(x,y) = 2 + \sin(x) + \cos(y)$ and $V_h^1(\Omega)$:

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
C = 1; B = 1; [e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 1; fem = fem_generator_Lagrange_2D_tri(mesh, degree);
a_name = @(x, y) 2 + sin(x) + cos(y); d_x = 0; d_y = 1; n_qp = 7;
M_yy = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem, d_x, d_y, ...
                fem, d_x, d_y, n_qp);

M_yy(23:27, 23:27) =
   (1,1)        3.218211137923407e+00
   (2,2)        3.227958957145486e+00
   (3,3)        3.237683978995185e+00
   (4,4)        3.247385230978409e+00
   (5,5)        3.257061742978031e+00
```

Example: The following script demonstrate how to generated a matrix on the domain $\Omega = (0,1) \times (0,1)$ for the function $a(x,y) = 2 + \sin(x) + \cos(y)$ and $V_h^1(\Omega)$ and $V_h^2(\Omega)$.

```
clear; mesh = mesh_generator_2D_tri(0, 1, 0, 1, 100, 100);
C = 1; B = 1; [e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree1 = 1; fem1 = fem_generator_Lagrange_2D_tri(mesh, degree1);
degree2 = 2; fem2 = fem_generator_Lagrange_2D_tri(mesh, degree2);
a_name = @(x, y) 2 + sin(x) + cos(y); d_x = 1; d_y = 0; n_qp = 7;
M_yy = FE_matrix_2D_Lagrange_tri(a_name, mesh, ...
                fem1, d_x, d_y, ...
                fem2, d_x, d_y, n_qp);
```

```
M_yy(23:27, 23:27) =
   (1,1)        1.072745609905088e+00
   (2,1)       -5.363729935850057e-01
   (1,2)       -5.379972745251984e-01
   (2,2)        1.075994942530611e+00
   (3,2)       -5.379976680054130e-01
   (2,3)       -5.396181331178422e-01
   (3,3)        1.079236675891843e+00
   (4,3)       -5.396185427740008e-01
   (3,4)       -5.412350300135296e-01
   (4,4)        1.082470485818186e+00
   (5,4)       -5.412354558046562e-01
   (4,5)       -5.428478035239135e-01
   (5,5)        1.085696048931339e+00
```

3.5, Approximation capability of $V_h^p(\Omega)$

Recall: for $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_d) \in \mathbb{N}_0^d$ with non-negative integers numbers $\alpha_i, i = 1, 2, \cdots, d$, we will use the following multi-index notations:

$$\begin{aligned} |\alpha| &= \alpha_1 + \alpha_2 + \cdots + \alpha_d, \quad \alpha! = \alpha_1! \alpha_2! \cdots \alpha_d! \\ X^\alpha &= x_1^{\alpha_1} \cdots x_d^{\alpha_d}, \ \forall X = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d, \quad D^\alpha = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} \end{aligned}$$

We will use norms of functions to describe the magnitude of a function. For a set $S \subseteq \Omega \subset \mathbb{R}^2$, let

$$\begin{aligned} L^2(S) &= \{u : S \to \mathbb{R} \mid \int_S (u(X))^2 dX < \infty\} \\ \|u\|_{0,S} &= \left(\int_S (u(X))^2 dX \right)^{1/2}, \quad \forall u \in L^2(S) \\ H^r(S) &= \left\{ u : S \to \mathbb{R} \mid D^\alpha u(X) \in L^2(S), 0 \le |\alpha| \le r \right\} \\ \|u\|_{r,S} &= \left(\sum_{0 \le |\alpha| \le r} \|D^\alpha u\|_{0,S}^2 \right)^{1/2}, \quad |u|_{r,S} = \left(\sum_{|\alpha| = r} \|D^\alpha u\|_{0,S}^2 \right)^{1/2} \end{aligned}$$

We often omit the subscript S when $S = \Omega$ or if no confusion will be caused by dropping the set S from the notation.

We now consider a simple approximation problem: given a function $u(X)$ with sufficient regularity, can we find/choose a finite element function $u_h(X) \in V_h^p(\Omega)$ such that $u_h(X)$ is a good approximation to $u(x)$ in a certain sense?

If the answer to this question is negative, i.e., none of the functions from the finite element space $V_h^p(\Omega)$ is a good approximation to $u(x)$, then, perhaps, we should not use this finite element space to solve a BVP where we need to find/compute/generate a finite element function from the chosen finite element space $V_h^p(\Omega)$ that can approximate the exact solution of the BVP which is often unknown.

We will use L^2 norm and H^1 norm to gauge the magnitude of the error function $u(x) - u_h(x)$ which means a finite element function $u_h(x)$ is a good approximation to $u(x)$ when $\|u - u_h\|_0$ and/or $\|u - u_h\|_1$ is small.

Given a function $u(X)$, we consider two usual finite element functions in $V_h^p(\Omega)$ that can be good approximations to $u(x)$. One is generated by interpolation in the finite element space, and the other is by the L^2 projection to the finite element space.

Finite element interpolation of $u(x)$ in $V_h^p(\Omega)$:

Assume $u(X)$ is continuous. The local finite element interpolation of $u(X)$ on each element $K \in \mathcal{T}_h$ is the polynomial $I_{h,K}u(x) \in V_h^p(K)$ defined by

$$I_{h,K}u(X) = \sum_{i=1}^{(p+1)(p+2)/2} u(X_{i,K})\phi_{i,p,K}(X) \tag{19}$$

where $X_{i,K} \in \mathcal{N}_{h,dof} \cap K, 1 \leq i \leq (p+1)(p+2)/2$, i.e., they are finite element nodes on the triangular element K.

The global finite element interpolation of $u(X)$ on Ω is a piecewise polynomial $I_h u(X)$ such that

$$I_h u(X)|_K = I_{h,K}u(X), \quad \forall K \in \mathcal{T}_h \tag{20}$$

It can be verified that

$$I_h u(X) = \sum_{X_i \in \mathcal{N}_{h,dof}} u(X_i)\phi_i(X) \text{ ???} \tag{21}$$

where ϕ_is are the basis functions spanning the finite element space $V_h^p(\Omega)$, i.e.,

$$V_h^p(\Omega) = span\{\phi_i, \ \forall X_i \in \mathcal{N}_{h,dof}\}$$

Error bounds for the finite element interpolation:

Theorem 1.5

(Local bounds for 2D FE interpolation error) Let K be an element in a mesh $\mathcal{T}_h$. Then, for every $u \in H^{k+1}(K) \cap C^0(\overline{K})$ with $0 \le k \le p$, its p-th degree FE interpolation $I_{h,K}u \in V_h^p(K)$ has the following error bound:

$$\|D^\alpha u - D^\alpha((I_{h,K}u))\|_{L^2(K)} \le C h_K^{k+1-r} |u|_{k+1,K}, \quad |\alpha| = r, \quad 0 \le r \le k \tag{22}$$

where h_K is the diameter of element K.

Theorem 1.6

(Global bounds for 2D FE interpolation error, Proposition 3.4 in BK1) For every $u \in H^{k+1}(\Omega) \cap C^0(\overline{\Omega})$ with $0 \le k \le p$, its p-th degree FE interpolation $I_h u \in V_h^p(\Omega)$ has the following error bound on on a family of shape regular (Definition 3.8 in BK1) meshes $\mathcal{T}_h$, $h > 0$ for the domain Ω:

$$\|D^\alpha u - D^\alpha(I_h u)\|_{L^2(\Omega)} \le C h^{k+1-r} |u|_{k+1,\Omega}, \quad |\alpha| = r, \quad 0 \le r \le k \tag{23}$$

In particular, for a function $u \in H^{p+1}(\Omega)$, we have

$$\begin{aligned} \|u - I_h u\|_{L^2(\Omega)} &\le C h^{p+1} |u|_{p+1,\Omega} \\ \left\|D^{(1,0)}u - D^{(1,0)}(I_h u)\right\|_{L^2(\Omega)} &\le C h^{p} |u|_{p+1,\Omega} \\ \left\|D^{(0,1)}u - D^{(0,1)}(I_h u)\right\|_{L^2(\Omega)} &\le C h^{p} |u|_{p+1,\Omega} \end{aligned}$$

The L^2 projection of $u(X)$ to $V_h^p(\Omega)$: Since $V_h^p(\Omega)$ is finite dimensional subspace of $L^2(\Omega)$, for any $u \in L^2(\Omega)$, we can defined its L^2 projection to V_h^p as the finite element function $P_h u \in V_h^p(\Omega)$ such that

$$\int_\Omega v_h(u - P_h u)dX = 0 \quad \text{or} \quad \int_\Omega v_h(P_h u)dX = \int_\Omega v_h u dX \quad \forall v_h \in V_h^p(\Omega) \tag{24}$$

Since $V_h^p(\Omega) = span\{\phi_i \mid X_i \in \mathcal{N}_{h,g}\}$, the L^2 project of u can be expressed as

$$P_h u(X) = \sum_{j=1}^{|\mathcal{N}_{h,g}|} u_j \phi_j(X)$$

with coefficients $u_j, j = 1, 2, \cdots, |\mathcal{N}_{h,g}|$ to be determined. Then, by the definition of L^2 projection, $P_h u$ must satisfy

$$\int_\Omega \phi_i \left(u - \sum_{j=1}^{|\mathcal{N}_{h,g}|} u_j \phi_j(X) \right) dX = 0 \quad \forall i = 1, 2, \cdots, |\mathcal{N}_{h,g}|$$

which leads to following equations for computing the coefficient u_j's:

$$\sum_{j=1}^{|\mathcal{N}_{h,g}|} \left(\int_\Omega \phi_i \phi_j dX \right) u_j = \int_\Omega \phi_i u dX, \ \ i = 1, 2, \cdots, |\mathcal{N}_{h,g}| \tag{25}$$

The equations above for the coefficient u_js in the L^2 projection $P_h u(X) = \sum_{j=1}^{|\mathcal{N}_{h,g}|} u_j \phi_j(X)$ can be put into the following matrix form:

$$M\vec{u} = \vec{b}, \;\; \vec{u} = (u_1, u_2, \cdots, u_{|\mathcal{N}_{h,g}|})^t$$

$$M = \left(\int_\Omega \phi_i \phi_j dx \right)_{i,j=1}^{|\mathcal{N}_{h,g}|}, \quad \vec{b} = \left(\int_\Omega \phi_i u dx \right)_{i=1}^{|\mathcal{N}_{h,g}|}$$

Since u and ϕ_j's are known, we use them to form the matrix M and the vector $\vec{b}$ and then solve $M\vec{u} = \vec{b}$ for $\vec{u}$. The linear independence of ϕ_j's guarantees that the matrix M is nonsingular; hence, it also guarantees a unique solution $\vec{u}$ or a unique L^2 projection $P_h u(X) = \sum_{j=1}^{|\mathcal{N}_{h,g}|} u_j \phi_j(X)$.

Error bounds for the L^2 projection to $V_h^p(\Omega)$:

Theorem 1.7

(Global bounds for 2D FE interpolation error) For every $u \in H^{k+1}(\Omega)$ with $0 \le k \le p$, its L^2 projection $P_h u \in V_h^p(\Omega)$ has the following error bound on Ω:

$$\|u - P_h u\|_{L^2(\Omega)} \le C h^{k+1} |u|_{k+1,\Omega} \tag{26}$$

Proof: ???

3.5.1, Evaluation of a finite element function in $V_h^p(\Omega)$

Recall that, on a computer, a FE function $u_h(X) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X) \in V_h^p(\Omega)$ on a domain Ω is described by the following objects:

- `mesh` for the mesh of Ω.
- `fem` for the finite element basis functions defined by the mesh.
- A function/program for the shape functions locally defined on each element of the mesh such as `shape_fun_2D_Lagrange_tri.m`
- An array `u_fe` for $\vec{u} = (u_j)_{j=1}^{|\mathcal{N}_{h,dof}|}$, i.e., the coefficients of the finite element functions $u_h(X)$.

Using a finite element function $u_h(X) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X)$ involves evaluating it at $X \in \Omega$ which, in principle, is usually done element by element in finite element methods for solving partial differential equations.

As before, we can implement a Matlab function to carry out the evaluation of a finite element function on a given/chosen element and this can be done by simply modifying the Matlab function `FE_evaluation_1D_Lagrange.m` developed for the 1D finite element space.

A Matlab function to evaluate a finite element function on an element:

```
function u = FE_evaluation_2D_Lagrange_tri(x, y, u_fe_loc, elem, ...
                                           degree, d_ind_x, d_ind_y)
u = zeros(size(x)); ldof = (degree + 1)*(degree + 2)/2;
for shape_index = 1:ldof
    u = u + u_fe_loc(shape_index)*shape_fun_2D_Lagrange_tri(x, y, ...
                    elem, degree, shape_index, d_ind_x, d_ind_y)
end
```

- `elem` is a 2×3 array providing the coordinates for the three vertices of the element on which the finite element function is to be evaluated. Note, `elem` for the k-th element in a mesh can be extracted by

  ```
  elem = mesh.p(:, mesh.t(:, k));
  ```

- `u_fe_loc` contains the coefficients of the finite element function at the finite element nodes on the element, and we can extract `u_fe_loc` from the coefficient array `u_fe` of $u_h(X)$ as follows:

  ```
  u_fe_loc = u_fe(fem.t(:, k));
  ```

- `x, y` are arrays for the x-y coordinates for the points in the element where $u_h(X)$ is to be evaluated.
- `d_ind_x, d_ind_y` are either 0 or 1 specifying the orders of the x and y partial derivative of $u_h(X)$ to be evaluated.
- `degee` takes the value p for the p-th degree finite element element function $u_h(X) \in V_h^p(\Omega)$.

A comparison of the 1D and 2D FE evaluation programs:

1D program:

```
function f = FE_evaluation_1D_Lagrange(x, u_fe_loc, elem, degree, d_index)
f = zeros(size(x));
for shape_index = 1:degree + 1
    f = f + u_fe_loc(shape_index)*shape_fun_1D_Lagrange(x, ...
                    elem, degree, shape_index, d_index);
end
```

2D program:

```
function u = FE_evaluation_2D_Lagrange_tri(x, y, u_fe_loc, elem, ...
                                           degree, d_ind_x, d_ind_y)
u = zeros(size(x)); ldof = (degree + 1)*(degree + 2)/2;
for shape_index = 1:ldof
    u = u + u_fe_loc(shape_index)*shape_fun_2D_Lagrange_tri(x, y, ...
                    elem, degree, shape_index, d_ind_x, d_ind_y)
end
```

Example: For $\Omega = [0,1]^2$, let $u(x,y) = \cos(x)\sin(y)$ and let $u_h(x,y) = I_h u(x,y)$ be its finite element interpolation in $V_h^2(\Omega)$ defined on a Cartesian mesh with $nx = 3, ny = 2$. Compute $u_h(x^*, y^*)$ where (x^*, y^*) is the center of the 10-th element in this mesh.

Recall that: $u_h(X) = I_h u(X) = \sum_{j=1}^{|\mathcal{N}_{h,dof}|} u_j \phi_j(X), \ u_i = \cos(x_i)\sin(y_i), \ \ (x_i, y_i) = X_i \in \mathcal{N}_{h,dof}$

```
u = @(x, y) cos(x).*sin(y); % function to be approximated
xmin = 0; xmax = 1; ymin = 0; ymax = 1; nx = 3; ny = 2;
mesh = mesh_generator_2D_tri(xmin, xmax, ymin, ymax, nx, ny);
C = 1; B = 1; [e, M_e, M_ne] = edges_in_mesh(mesh, C, B);
mesh.e = e; mesh.M_ne = M_ne;
degree = 2; fem = fem_generator_Lagrange_2D_tri(mesh, degree);
u_fe = cos(fem.p(1,:)).*sin(fem.p(2,:)); % coeff for the FE interpolation
k = 10; elem = mesh.p(:, mesh.t(:, k)); % k-th element
xs = (1/3)*sum(elem(1, :)); ys = (1/3)*sum(elem(2, :));
% evaluate the FE function at Xs = (xs, ys)
u_fe_loc = u_fe(fem.t(:, k));
Ihus = FE_evaluation_2D_Lagrange_tri(xs, ys, u_fe_loc, elem, ...
        degree, 0, 0);
fprintf('Ihus = %.15e, u(xs, ys) = %.15e\n', Ihus, u(xs, ys))
```

The above script produces `uc = 5.252635283504101e-01` which seems to be a good approximate to $u(x^*, y^*) = \cos(x^*)\sin(y^*) = 5.253716613258217e-01$.