Notes ⋆ Analyse Numérique - Part II: Implementation of the Finite Elements ⋆ Automatic Meshes for Polygonal Structures

Nguyen Quan Ba Hong*

October 30, 2018

Abstract

This context includes the material given by Prof. Éric Darrigrand in the course *Finite Element Method*, which is taught by Prof. Nicolas Seguin and Prof. Éric Darrigrand in the Master 2 Fundamental Mathematics program 2018-2019, with my MATLAB scripts provided.

Brief introduction. "This lecture is a numerical counterpart to *Sobolev spaces & elliptic equations*. In the first part, after some reminders on linear elliptic partial differential equations, the approximation of the associated solutions by the finite element methods is investigated. Their construction and their analysis are described in one and two dimensions. The second part of the lectures consists in defining a generic strategy for the implementation of the method based on the variational formulation. A program is written in MATLAB (implementable with MATLAB or OCTAVE)."

*Master 2 student at UFR mathématiques, Université de Rennes 1, Beaulieu - Bâtiment 22 et 23, 263 avenue du Général Leclerc, 35042 Rennes CEDEX, France.
E-mail: nguyenquanbahong@gmail.com
Blog: www.nguyenquanbahong.com

Contents

1 Structure of a Mesh

A mesh of triangles will be represented by two tables:

- the table of coordinates of the points, of size $NbPoints \times 2$ (i.e., Number of Points×2)

$$\text{coords}\,(l, L) = L^e \text{ coordinate of the point number } l. \tag{1.1}$$

- the table of triangles, of size $NbTriangles \times NbNodes$ (i.e., Number of Triangles×Number of Nodes)

$$\text{triangles}\,(k, i) = \text{number of the } i^{\text{th}} \text{ node of the triangle number } k. \tag{1.2}$$

For triangles $\mathbb{P}_1$, $NbNodes = 3$; for triangles $\mathbb{P}_2$, $NbNodes = 6$.

In MATLAB, we use the structure of data `struct`, similar to the concept of objects of an object-oriented language like C++. We use the following constructor

```
function mesh_s = mesh_new(name, coords, triangles)
% define a new mesh_structure
% Author: G. Vial
% Date: 08/09/2010
% Last update: 22/09/11 -- E. Darrigrand
% usage: mesh_s = mesh_new(name,coords,triangles)
% input -
%    name : string
%    coords : (NbPts x 2) array
%    triangles : (NbTriangles x NbNodes) array
% output -
%    mesh_s : mesh structure

mesh_s = struct('name', name, 'coords', coords, 'triangles', triangles);
```

The access to the components is done as follows

```
coords = [0,0;1,0;1,1;0,1;.5,.5];
triangles = [1 2 5;2 3 5;5 4 3;1 5 4];
my_mesh = mesh_new(' ', coords, triangles);
co = getfield(my_mesh, 'coords');
tr = getfield(my_mesh, 'triangles');
my_mesh.name = 'Example of a mesh';
```

The previous example corresponds to the mesh of the unit square with four identical triangles as defined by Fig. 1 (the red numbers correspond to the numbering of the triangles, the boldface numbers are the global numbers of the points of the mesh).

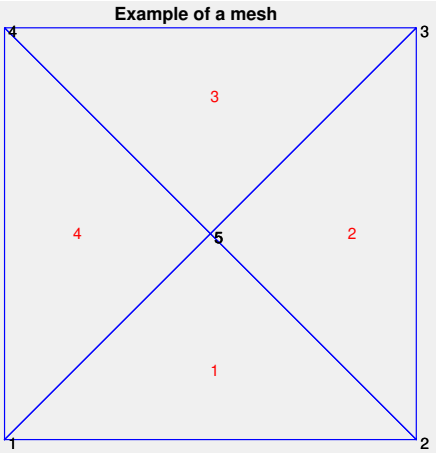

Figure 1: Mesh of the unit square with 4 triangles $\mathbb{P}_1$.

The tables `coords` and `triangles` define fully the mesh. However, it may be useful to use other tools. The following gives information on the edges of the mesh:

- `edges`$(k,:)$ = numbers of the points that determine the edge numbered k,
- `edges_triangles`$\{k\}$ = couples (element, local number of edge) for the elements which contain the edge number k.

The following example precises the structure of this information: let us suppose that the edge numbered 2 is the edge that connects the points 2 and 5. Since it is shared by the elements 1 and 2, for which it is the edge respectively locally numbered 2 and 3, we have

`edges(2,:) = [2,5]; edges_triangles{2} = {(1,2),(2,3)}.`

The computation with MATLAB uses a *cell-array*, which is more convenient than a table:

```
edges = [1,2;2,5;1,5;2,3;3,5;4,5;3,4;1,4];
edges_triangles{1} = [1,1];
edges_triangles{2} = [1,2;2,3];
edges_triangles{3} = [1,3;4,1];
edges_triangles{4} = [2,1];
edges_triangles{5} = [2,2;3,3];
edges_triangles{6} = [3,1;4,2];
edges_triangles{7} = [3,2];
edges_triangles{8} = [4,3];
```

Problem 1.1. *Write a* MATLAB *function which builds the tables* `edges` *and* `edges_triangles`*, the head of which is*

```
function [edges, edges_triangle] = build_edge_connectivity(mesh)
```

Solution. To build the table `edges`, we first initialize it as an 'empty matrix'. We then loop on all the triangles of the given mesh. For each triangle, we compare[1] its three edges with all edges which is currently stored in the table `edges`. There are only two possibilities.

- The first possibility is that the considered edge has not been stored in the table `edges` yet. In this case, we store that edge as a new row of the table `edges`.
- The second possibility is that the edge considered was stored in the table `edges` already. In this case, we skip it and consider the next edge (or the next triangle in the loop, if all three edges of the current triangle have been considered).

After the table `edges` is built successfully, we construct the table `edges_triangles`. To do this, we loop on all the edges in the table `edges`. For each edge, we make another loop on all the triangles of the mesh given. For each triangle in the second loop, we compare its three edges with the edge considered in the first loop. If there is a coincidence, we mark it on the current entry of the table `edges_triangles` by the convention of local numbering of the reference triangle $\mathbb{P}_1$ described above.

Here is my MATLAB script:

```
function [edges, edges_triangles] = build_edge_connectivity(mesh)
% Build the tables edges & edges_triangles of a mesh
% Author: Nguyen Quan Ba Hong
% Date: 06/10/2018
% Last Update: 16/10/2018
% Input:
% + mesh: a structured mesh
% Outputs:
% + edges: edges(K,:) = numbers of the points that determining the edge
% numbered K
% + edges_triangles{K} = couples(element, local number of edge) for the
% elements which contain the edge number K

%% Build Table edges
edges = [];
for K = 1:size(mesh.triangles,1) % Loop on Triangles of the Mesh
    % Read the Edges of the K-th Triangle
    temp = sort([mesh.triangles(K,[1,2]);
        mesh.triangles(K,[2,3]);
        mesh.triangles(K,[1,3])],2);
    flag = ones(3,1);
    for i = 1:3 % Loop on 3 Edges of the K-th Triangle
```

[1] To compare easier, each edge of the considered triangle should be sorted first.

```
        for j = 1:size(edges,1)
            % Check if the edge considered is already in table edges or not
            if (edges(j,:) == temp(i,:))
                flag(i) = 0;
            end
        end
        if (flag(i) == 1)
            edges = [edges; temp(i,:)];
        end
    end
end

%% Build Cell-Array edges_triangles
for i = 1:size(edges,1) % Loop on the Edges
    edges_triangles{i} = [];
    for j = 1:size(mesh.triangles,1) % Loop on the Triangles of the Mesh
        if (edges(i,:) == sort(mesh.triangles(j,[1,2])))
            edges_triangles{i} = [edges_triangles{i}; j,1];
        elseif (edges(i,:) == sort(mesh.triangles(j,[2,3])))
            edges_triangles{i} = [edges_triangles{i}; j,2];
        elseif (edges(i,:) == sort(mesh.triangles(j,[1,3])))
            edges_triangles{i} = [edges_triangles{i}; j,3];
        end
    end
end
```

For illustrations, running this script in the main function as

```
[edges,edges_triangles] = build_edge_connectivity(my_mesh);
for i = 1:size(edges,1)
   edges_triangles{i}
end
```

gives us exactly the `edges` and `edges_triangles` as above. □

Problem 1.2. *Write a* MATLAB *function the head of which is*

```
function mesh_P2 = P1_to_P2(mesh_P1)
```

and which builds a $\mathbb{P}_2$ mesh structure from a given $\mathbb{P}_1$ mesh structure. The local numbering should follow the convention of local numbering of the reference triangles $\mathbb{P}_2$ as follows: the midpoints of the edges 1, 2, 3 are labeled 4, 5, 6, respectively.

Solution. To build a $\mathbb{P}_2$ mesh structure from a given $\mathbb{P}_1$ mesh structure, it suffices to modify the matrices `coords` and `triangles` of the old (i.e., $\mathbb{P}_1$) mesh structure.

The new `coords` is modified easily by adding the midpoints of all edges of the $\mathbb{P}_1$ mesh. To do this, we first call the function `build_edge_connectivity` of Problem 1.1 to obtain the table `edges`. Because the table `edges_triangles` is not needed in this problem, so we replace this output with `~` to save memory when the function `build_edge_connectivity` is called, see the script below. We then loop on all edges of the $\mathbb{P}_1$ mesh structure[2]. For each edge in the loop, we add its midpoint as a new row in the table `coords`. An important notice is that the midpoint of the i^{th} edge in the $\mathbb{P}_1$ will be stored in the $(i + \text{the number of nodes of } \mathbb{P}_1 \text{ mesh structure})^{\text{th}}$ row in the table `coords` being updated. This notice is very useful in the below construction of the table `triangles` for the new mesh structure.

The size of the table `triangles` of the $\mathbb{P}_1$ mesh structure is the number of triangles of $\mathbb{P}_1 \times 3$, whereas the size of the table `triangles` of the $\mathbb{P}_2$ mesh structure is the number of triangles of $\mathbb{P}_1 \times 6$. The last three columns, i.e., 4, 5, 6, are added to store the midpoints of three edges of each triangle of the $\mathbb{P}_1$ mesh structure.

Here is my MATLAB script:

```
function mesh_P2 = P1_to_P2(mesh_P1)
% Build a P_2 mesh structure from a given P_1 mesh structure. The local
% numbering should follow the convention of Local numbering of the
% reference triangles P_2 as follows: the midpoints of the edges 1, 2, 3
% are labeled 4, 5, 6, respectively.
% Author: Nguyen Quan Ba Hong
% Date: 16/10/2018
% Last Update: 16/10/2018
% Input:
% + mesh_P1: a P_1-structured mesh
% Output:
% + mesh_P2: a P_2-structured mesh

%% Build edges & edges_triangles
[edges, ~] = build_edge_connectivity(mesh_P1);

%% Construct coords of mesh_P2
coords = mesh_P1.coords; % Utilize the Nodes of the Old Mesh
for i = 1:size(edges,1) % Loop on the Edges of mesh_P1
    % Add the Midpoint of i-th Edge of mesh_P1 to coords of mesh_P2
    coords = [coords; (coords(edges(i,1),:) + coords(edges(i,2),:))/2];
end
```

[2] We should not loop on all triangles of the $\mathbb{P}_1$ mesh since all interior edges will be considered twice, and so are their midpoints.

```
%% Construct Triangles of mesh_P2
triangles = zeros(size(mesh_P1.triangles,1),6);
triangles(:,[1,2,3]) = mesh_P1.triangles; % Utilize the Triangles of mesh_P1
for K = 1:size(mesh_P1.triangles,1) % Loop on Triangles of mesh_P1
    for i = 1:size(edges,1)
        if (sort(mesh_P1.triangles(K,[1,2])) == edges(i,:))
            triangles(K,4) = size(mesh_P1.coords,1) + i;
        elseif (sort(mesh_P1.triangles(K,[2,3])) == edges(i,:))
            triangles(K,5) = size(mesh_P1.coords,1) + i;
        elseif (sort(mesh_P1.triangles(K,[1,3])) == edges(i,:))
            triangles(K,6) = size(mesh_P1.coords,1) + i;
        end
    end
end

%% Build a P_2-Structured Mesh
mesh_P2 = mesh_new('Mesh_P2', coords, triangles);
```

For illustrations, we run the following commands

```
mesh_P2 = P1_to_P2(my_mesh);
figure(1)
mesh_plot(mesh_P2,1)
```

which produce the following figure

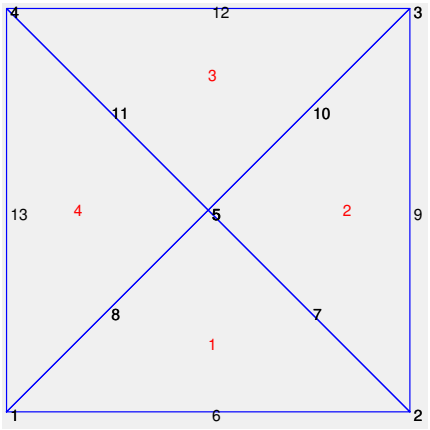

Figure 2: A $\mathbb{P}_2$ mesh structure.

This completes our solution. □

In the annex, we give the code of the function `mesh_plot`, which offers a view of the mesh structure ($\mathbb{P}_1$ or $\mathbb{P}_2$).

2 Triangulation of Delaunay

The MATLAB function `delaunay` builds a triangulation of Delaunay based on a set of points. If the table `coords`, of size $NbPoints \times 2$ contains the coordinates of the points of such a set, the call of the function is done with the following options

```
triangles = delaunay(coords(:,1),coords(:,2),{'Qt','Qbc','Qc','Qz'});
```

However, when I run this command in my MATLAB R2015a, the MATLAB's Command Window gives the following error

```
Error using delaunay
DELAUNAY no longer supports or requires Qhull-specific options.
 Please remove these options when calling DELAUNAY.
```

And, the reduced command

```
triangles = delaunay(coords(:,1),coords(:,2));
```

works well. So, we use this replacement to carry out the Delaunay's triangulation.

If the function is called with the table `coords` of the example of Fig. 1, one obtains the same triangulation with maybe different order in the numbering. Indeed,

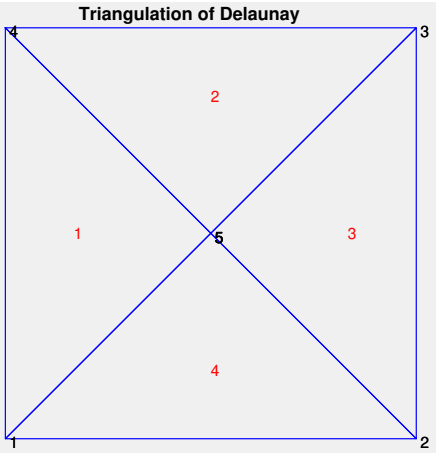

Figure 3: The triangulation of Delaunay has different orders in the numbering, Cf. Fig. 1.

Problem 2.1. *Write a* MATLAB *function the head of which is*

```
function [circumcenter, circumradius] = circumcircle(triangle_coor)
```

and which computes the center of the circumscribed circle and its radius, for a triangle given through the coordinates of its vertices. The table `triangle_coor` *is of size* 3×2.

Using this function, plot the circumscribed circles of the triangles obtained with the function `delaunay` *and check that no open disk contains any point of the triangulation except the nodes of the mesh.*

Solution. From the given coordinates of the vertices, we can easily compute the length of three edges and the area of that triangle. To compute the area S, there are two ways. The first one is to use the MATLAB build-in function `polyarea`. The second one is to use the Heron's formula

$$S = \frac{1}{4}(a+b+c)(b+c-a)(c+a-b)(a+b-c). \tag{2.1}$$

To compute the radius of the circumscribed circle of the considered triangle, we use the formula

$$R = \frac{abc}{4S}. \tag{2.2}$$

To compute the coordinates of the center of the circumscribed circle of the considered triangle, we use the following formula

$$O = \frac{a^2\left(b^2+c^2-a^2\right)A + b^2\left(c^2+a^2-b^2\right)B + c^2\left(a^2+b^2-c^2\right)C}{2\left(b^2c^2+c^2a^2+a^2b^2\right) - \left(a^4+b^4+c^4\right)}, \tag{2.3}$$

where A, B, and C are the coordinates given, and O denotes the coordinates of the center of the circumscribed circle of the triangle ABC.

Here is my MATLAB script:

```
function [circumcenter, circumradius] = circumcircle(triangle_coor)
% Compute the center of the circumscribed circle and its radius, for a
% triangle given through the coordinates of its vertices
% Author: Nguyen Quan Ba Hong
% Date: 06/10/2018
% Last Update: 16/10/2018
% Input:
% + triangle_coor: is of size 3x2, gives the coordinates of three vertices
% of a triangle
% Outputs:
% + circumcenter: the center of the circumscribed circle of that triangle
% + circumradius: the radius of the circumscribed circle of that triangle

%% Compute Area of the Triangle
S = polyarea(triangle_coor(:,1), triangle_coor(:,2));

%% Compute Length of Edges of the Triangle
```

```
a = norm(triangle_coor(2,:) - triangle_coor(3,:)); % Length of edge BC
b = norm(triangle_coor(3,:) - triangle_coor(1,:)); % Length of edge CA
c = norm(triangle_coor(1,:) - triangle_coor(2,:)); % Length of edge AB

%% Alternative Way to Compute Area
% S = 1/4*sqrt((a+b+c)*(a+b-c)*(b+c-a)*(c+a-b)); % Heron's Formula

%% Compute Circumradius
circumradius = 1/4*a*b*c/S;

%% Compute Coordinates of Circumcenter
temp = [a^2*(b^2 + c^2 - a^2), b^2*(c^2 + a^2 - b^2), c^2*(a^2 + b^2 - c^2)];
circumcenter = temp(1)*triangle_coor(1,:) + temp(2)*triangle_coor(2,:) ...
    + temp(3)*triangle_coor(3,:);
circumcenter = circumcenter/sum(temp);
```

Running following commands in the main script with the above meshes

```
my_mesh.name = 'Circumscribed Circles of Triangles';
figure
hold on
mesh_plot(my_mesh,0)
for i = 1:size(my_mesh.triangles,1)
    [center, radius] = circumcircle(my_mesh.coords(my_mesh.triangles(i,:),:));
    t = linspace(0,2*pi);
    x = center(1) + radius*cos(t);
    y = center(2) + radius*sin(t);
    plot(x,y);
end
axis equal
```

gives us the following figure

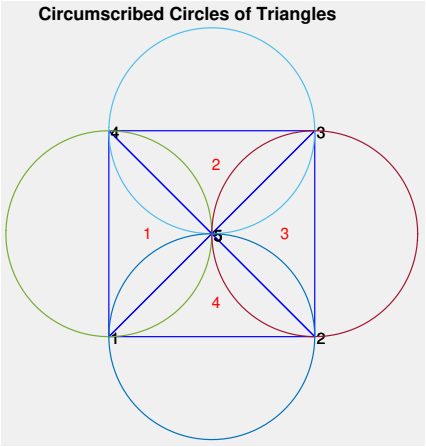

Figure 4: Circumscribed circles of the triangles obtained with the function `delaunay`.

It is clear that these circumscribed circles do not intersect any point of the triangulation except the nodes of the mesh. □

Remark 2.1. *Here is an alternative* MATLAB *script of Prof. Éric Darrigrand (see* [**1**]*):*

```
function [circumcenter,circumradius] = circumcircle(triangle_coor)
% Compute the circumscribed circle data of a triangle given
% by the coordinates of its vertices.
% Author: E.Darrigrand
% Date: 22/09/2011
% Last update: 22/09/2011
%
% usage: [circumcenter,circumradius] = circumcircle(triangle_coor)
%
% input -
%   triangle_coor : 3x2 array containing the coordinates of the vertices of the triangle.
% output -
%   circumcenter : center of the circumscribed circle
%   circumradius : radius of the circumscribed circle
%
a = triangle_coor(2,1) - triangle_coor(1,1);
b = triangle_coor(2,2) - triangle_coor(1,2);
c = triangle_coor(3,1) - triangle_coor(1,1);
d = triangle_coor(3,2) - triangle_coor(1,2);
e = 0.5 * (triangle_coor(2,1) + triangle_coor(1,1));
f = 0.5 * (triangle_coor(2,2) + triangle_coor(1,2));
g = 0.5 * (triangle_coor(3,1) + triangle_coor(1,1));
h = 0.5 * (triangle_coor(3,2) + triangle_coor(1,2));
```

```
%
alpha = a*e + b*f;
beta = c*g + d*h;
%
% With the notations above, the center has the coordinates (x,y) given below:
%
y = (alpha*c - a*beta)/(b*c - a*d);
if (abs(a)>abs(c))
   x = (alpha - b*y)/a;
else
   x = (beta - d*y)/c;
end
circumcenter = [x,y];
%
% The radius is then the distance between the center and a vertex:
%
vect = [x,y]-triangle_coor(1,:);
circumradius = sqrt(vect*vect');
```

3 Automatic Mesh

If you have a triangulation from a set of points defined on the boundary of a domain of computation, to apply the finite elements you will require some points in the interior of the domain. The efficient methods are not easy to implement. Here, we consider a very simple way to implement a strategy that introduces points inside the domain of computation.

The principle is the following: we consider a triangulation $\mathcal{T}_n$ given at a step n. We determine the triangle $K \in \mathcal{T}_n$ which is identified as the "worst" (i.e., of greatest edge), and we denote G its iso-barycenter. The new triangulation $\mathcal{T}_{n+1}$ is built by the insert of G into the set of points of $\mathcal{T}_n$.

For the initialization of the process, we recommend the consideration of the triangulation $\mathcal{T}_0$ obtained from a subdivision of the polygonal boundary of the domain of computation. This subdivision would be a set of segments of size around h, where h is the requested size for the elements of the mesh.

We mention again that this method is not efficient. Many different improvements can be done, for example, inserting the center of the circumscribed circle instead of its iso-barycenter (except if it is outside of the domain, in that case, we insert the midpoint of the largest edge).

Problem 3.1. *Write a* MATLAB *function which determines the "worst" element of a given mesh. Its head would be*

```
function [triangle_number, max_edge_size] = find_worst_triangle(mesh)
```

Then, write a function which builds the mesh according to the described strategy, from a subdivision of the boundary of the domain of computation:

```
function mesh_out = mesh_by_barycenters(boundary_vertices, mesh_size)
```

They `mesh_size` *is a scalar which specify the required maximal size for the elements of the output mesh* `mesh_out`*. The vector* `boundary_vertices` *gives the coordinates of the vertices of the polygonal boundary of the computational domain.*

Solution. Here is my MATLAB script for the first task:

```
function [triangle_number, max_edge_size] = find_worst_triangle(mesh)
% Determine the "worst" element of a given mesh, i.e., the element has the
% longest edge in the mesh, and provide the length of that longest edge
% Author: Nguyen Quan Ba Hong
% Date: 29/10/2018
% Last Update: 29/10/2018
% Input:
% + mesh: a mesh structure
% Outputs:
% + triangle_number: Index of the "worst" element of the mesh given
% + max_edge_size: the length of the longest edge in the mesh given

%% Utilize the Tables edges & edges_triangles
[edges, edges_triangles] = build_edge_connectivity(mesh);

%% Compute the Lengths of All Edges in the Mesh Given
length_edges = zeros(size(edges,1),1);
for i = 1:size(edges,1) % Loop on All Edges of the Mesh Given
    % Compute the Length of the i-th Edge
    length_edges(i) = norm(mesh.coords(edges(i,1),:) - mesh.coords(edges(i,2),:));
end

%% Determine the Longest Edge and Its Index
[max_edge_size, edge_number] = max(length_edges);

%% Determine a Triangle Containing the Longest Edge
triangle_number = edges_triangles{edge_number}(1,1);
```

And here is my MATLAB script for the second task:

```
function mesh_out = mesh_by_barycenters(boundary_vertices, mesh_size)
% Build the mesh according to the following strategy:
```

```
% 1) For the initialization of the process: consider the triangulation
% T_0 obtained from a subdivision of the polygonal boundary of the domain
% of computation. This subdivison would be set of segments of size around
% mesh_size, where mesh_size is the requested size for the elements of the
% mesh.
% 2) Principle: Consider a triangulation T_n given at a step n, determine
% the "worst" triangle (i.e., of greatest edge), denote G its
% iso-barycenter. The new triangulation T_{n+1} is built by the insert of G
% into the set of points of T_n
% Author: Nguyen Quan Ba Hong
% Date: 08/10/2018
% Last Update: 29/10/2018
% Inputs:
% + boundary_vertices: vector which gives the coordinates of the
% vertices of the polygonal boundary of the computational domain
% + mesh_size: a scalar which specify the required maximal size for the
% elements of the output mesh
% Output:
% + mesh_out: the mesh built according to the described strategy, from a
% subdivision of the boundary of the domain of computation

%% Initialization of the Process: Zeroth Triangulation
coords = boundary_vertices; % Initialzie
for i = 1:size(boundary_vertices,1)-1 % Loop on All Non-Last Boundary Edges
    % Compute the Length of the i-th Boundary Edge
    length = norm(boundary_vertices(i,:) - boundary_vertices(i+1,:));
    loop = 2*ceil(length/mesh_size); % Number of Points for Partitioning
    for j = 1:loop-1
        % Equally-Spaced Partition of the i-the Boundary Edge
        coords = [coords; (loop-j)/loop*boundary_vertices(i,:) ...
            + j/loop*boundary_vertices(i+1,:)];
    end
end
% Compute the Length of the Last Boundary Edge
length = norm(boundary_vertices(i,:) - boundary_vertices(1,:));
loop = 2*ceil(length/mesh_size); % Number of Points for Partitioning
for j = 1:loop-1
    % Equally-Spaced Partition of the Last Boundary Edge
    coords = [coords; (loop-j)/loop*boundary_vertices(i,:) ...
        + j/loop*boundary_vertices(1,:)];
end
```

```
%% Perform Delaunay Triangulation of the New Table coords
triangles = delaunay(coords(:,1), coords(:,2));

%% Modify the Triangulation by Adding Iso-Barycenter
tol = 1e-2; % Tolerence for the criteria of degenerate triangles
while 1
    mesh_out = mesh_new('', coords, triangles); % Update Mesh Structure
    mesh_out = delete_degenerate_triangles(mesh_out, tol);
    % Find the Worst Element and Its Radius in the Current Mesh
    [triangle_number, max_edge_size] = find_worst_triangle(mesh_out);
    if (max_edge_size <= mesh_size)  % Stopping Criteria
        break % the Current Mesh Fulfills the Requirements on Size
    end
    % Add Iso-barycenter of the Worst Triangle to the Mesh
    coords = [coords; 1/3*sum(coords(mesh_out.triangles(triangle_number,:),:))];
    triangles = delaunay(coords(:,1), coords(:,2));
end
```

This completes our solution. □

4 Quality of the Mesh

In the error estimates of the finite-element method, the *roundness* of the elements of the mesh is significantly involved. It is given by the ratio between the radii of the circumscribed and inscribed circles:

$$\sigma(K) := \frac{h(K)}{\rho(K)}. \tag{4.1}$$

Problem 4.1. *Write a* MATLAB *function which computes the roundness of a triangle. Its head would be*

```
function sigma = roundness(triangle_coor)
```

Then, write a function which computes the quality of a mesh, defined as the largest roundness of the elements:

```
function quality = mesh_quality(mesh)
```

Solution. Here is my MATLAB scripts:

```
function sigma = roundness(triangle_coor)
% Compute the roundness of a triangle, which is given by the ratio between
% the radii of the circumscribed and the inscribed circles
```

```
% Author: Nguyen Quan Ba Hong
% Date: 09/10/2018
% Last Update: 17/10/2018
% Input:
% + triangle_coor: is of size 3x2, gives the coordinates of three vertices
% of a triangle
% Output:
% + sigma: the roundness of the triangle given

%% Compute the Radius of the Circumscribed Circle of the Triangle
[~, circumradius] = circumcircle(triangle_coor);

%% Compute the Radius of the Inscribed Circle of the Triangle
[~, inscribed_radius] = inscribed_circle(triangle_coor);

%% Compute the Roundness of the Triangle
sigma = circumradius/inscribed_radius;
```

and

```
function quality = mesh_quality(mesh)
% Compute the quality of a given mesh, defined as the largest roundness of the
% elements of that mesh
% Author: Nguyen Quan Ba Hong
% Date: 09/10/2018
% Last Update: 17/10/2018
% Inpute:
% + mesh: a mesh structure
% Output:
% + quality: the quality of the mesh given

%% Store the Roundness of All Triangles in the Mesh
sigma = zeros(size(mesh.triangles,1),1);
for i = 1:size(mesh.triangles,1) % Loop on All Triangles of the Mesh
    % Compute the Roundness of i-th Triangle in the Mesh
    sigma(i) = roundness(mesh.coords(mesh.triangles(i,:),:));
end

%% Compute the Quality of the Given Mesh
quality = max(sigma);
```

This completes our solution. □

Numerous strategies exist to improve the quality of a given mesh. Here, we propose to implement the most accessible: each interior point of the mesh (i.e., not on the boundary) is replaced by the iso-barycenter of its neighbors.

Problem 4.2. *Write a* MATLAB *function which optimizes a mesh following the method described just above:*

```
function mesh_out = mesh_optimize(mesh_in)
```

(you may need a function which determines if a point is interior or not; you may use the list `edges_triangles` *built in Problem 1.1).*

Solution.
```
function mesh_out = mesh_optimize(mesh_in)
% Optimize a mesh by the following strategy: Each interior point of the
% mesh (i.e. not on the boundary) is replaced by the iso-barycenter of its
% neighbors
% Author: Nguyen Quan Ba Hong
% Date: 08/10/2018
% Last Update: 17/10/2018
% Input:
% + mesh_in: a mesh structure needing improving the quality
% Output:
% + mesh_out: optimized mesh structure

%% Reuse All the Nodes on the Boundary of the Mesh Given
boundary_nodes = boundarynodes(mesh_in); % Initialize
coords = boundary_nodes; % Storage of Nodes of the New Mesh

%% Add Iso-Barycenters of All Triangles of the Mesh Given
for i = 1:size(mesh_in.triangles,1) % Loop on All Triangles of the Mesh Given
    coords = [coords; 1/3*sum(mesh_in.coords(mesh_in.triangles(i,:),:))];
end

%% Create the Final Mesh
triangles = delaunay(coords(:,1), coords(:,2)); % Delaunay Triangulation
mesh = mesh_new('', coords, triangles);
mesh_out = delete_degenerate_triangles(mesh, 1e-2);
```

□

A Appendices: Matlab Scripts

This section consists of all supplementary needed

1.1 Delete All Degenerate Triangles in a Mesh

```
function mesh_out = delete_degenerate_triangles(mesh, tol)
% Delete all degenerate triangles in the mesh given. Define a "degenerate"
% triangle as a triangle whose the radius of its corresponding inscribed
% circle is less than some given tolerance, e.g. inscribed_radius < 1e-3
% Author: Nguyen Quan Ba Hong
% Date: 08/10/2018
% Last Update: 29/10/2018
% Input:
% + mesh: a mesh structure
% Output:
% + mesh_out: a mesh structure without degenerate triangles

%% Classify Degenerate and Nondegenerate Triangles
triangles = []; % Initialize
for i = 1:size(mesh.triangles,1) % Loop on All Triangles of the Mesh Given
    % Compute the Inscribed Radius of the i-th Triangle
    [~, inscribed_radius] = inscribed_circle(mesh.coords(mesh.triangles(i,:),:));
    if (inscribed_radius > tol)
         triangles = [triangles; mesh.triangles(i,:)];
    end
end

%% Create a New Mesh without Degenerate Triangles
mesh_out = mesh_new('', mesh.coords, triangles);
```

1.2 Mesh Improvement

1.3 Mesh Refinement

Here is my script:

```
function mesh_P2 = P1_to_P2(mesh_P1)
% Refine a mesh
% Author: Nguyen Quan Ba Hong
% Date: 06/10/2018
% Last Update: 08/10/2018

[edges,edges_triangles] = build_edge_connectivity(mesh_P1);

%% Construct coords of mesh_P2
```

```
coords = mesh_P1.coords; % Utilize the nodes of the old mesh
for i = 1:size(edges,1)
    % Add the midpoint of i-th edge of mesh_P1
    coords = [coords;(coords(edges(i,1),:) + coords(edges(i,2),:))/2];
end

%% Construct triangles of mesh_P2
triangles = []; % Initialize triangles of mesh_P2
for i = 1:size(mesh_P1.triangles,1)
    for j = 1:size(edges,1)
        if (edges(j,:) == sort([mesh_P1.triangles(i,1),mesh_P1.triangles(i,2)]))
            midpoint_edge1 = j + size(mesh_P1.coords,1);
        end
        if (edges(j,:) == sort([mesh_P1.triangles(i,2),mesh_P1.triangles(i,3)]))
            midpoint_edge2 = j + size(mesh_P1.coords,1);
        end
        if (edges(j,:) == sort([mesh_P1.triangles(i,1),mesh_P1.triangles(i,3)]))
            midpoint_edge3 = j + size(mesh_P1.coords,1);
        end
    end
    triangles = [triangles;mesh_P1.triangles(i,1),midpoint_edge1,midpoint_edge3;
        mesh_P1.triangles(i,2),midpoint_edge2,midpoint_edge1;
        mesh_P1.triangles(i,3),midpoint_edge3,midpoint_edge2;
        midpoint_edge1,midpoint_edge2,midpoint_edge3];
end
mesh_P2 = mesh_new('Mesh',coords,triangles);
```

To implement how this subscript works, we run the following commands in the main script

```
for i = 1:5
    my_mesh = P1_to_P2(my_mesh);
    figure(i)
    mesh_plot(my_mesh,0)
    axis equal
end
```

to obtain

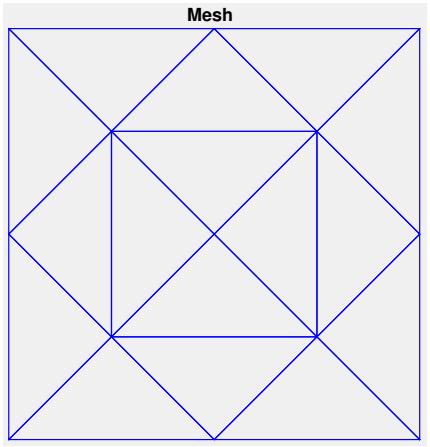

Figure 5: Apply the script `P1_to_P2` once without showing numbers.

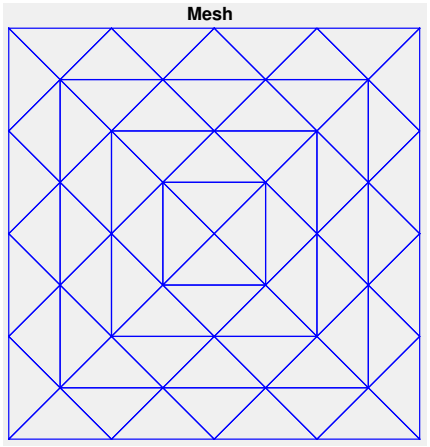

Figure 6: Apply the script `P1_to_P2` twice without showing numbers.

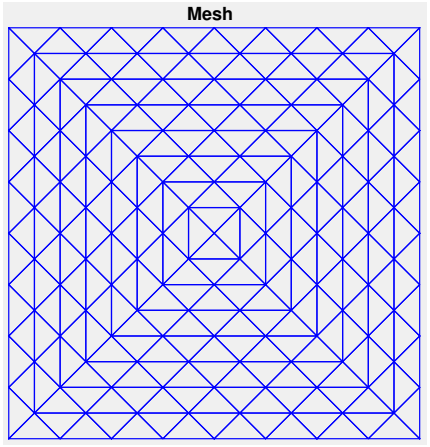

Figure 7: Apply the script `P1_to_P2` three times without showing numbers.

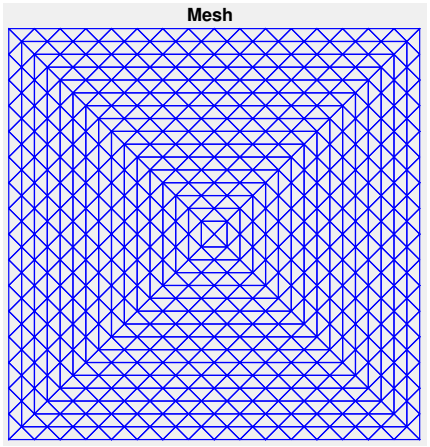

Figure 8: Apply the script `P1_to_P2` four times without showing numbers.

and with the command `mesh_plot(my_mesh,1)`,

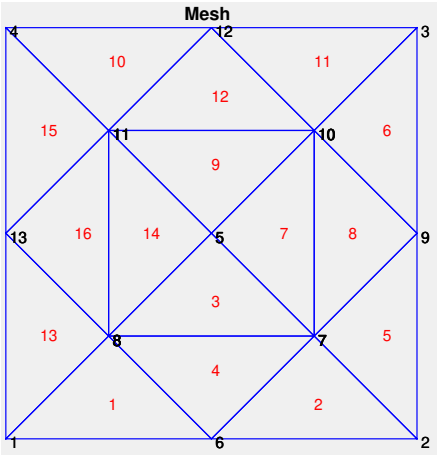

Figure 9: Apply the script `P1_to_P2` four times with showing numbers.

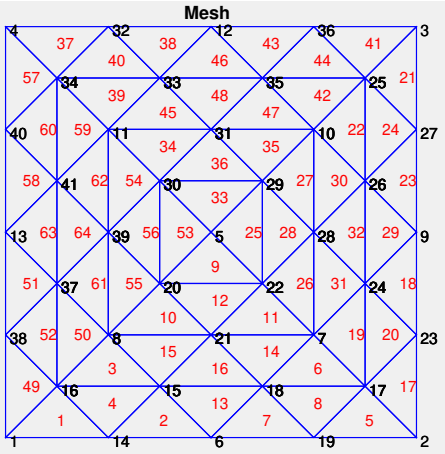

Figure 10: Apply the script `P1_to_P2` four times with showing numbers.

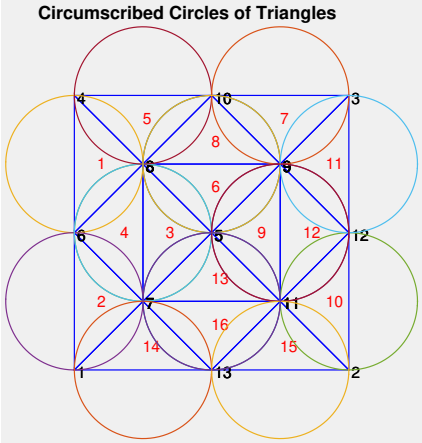

Figure 11: Circumscribed circles of the triangles obtained with the function `delaunay`.

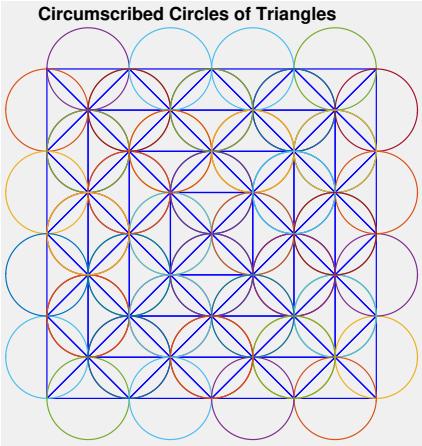

Figure 12: Circumscribed circles of the triangles obtained with the function `delaunay`.

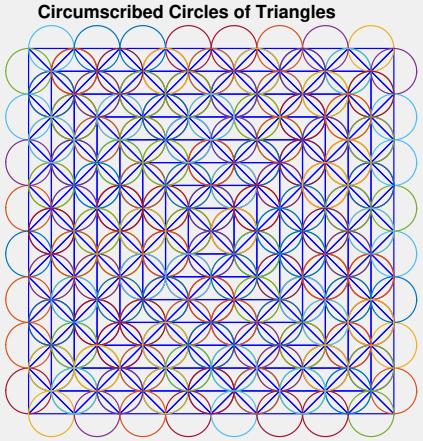

Figure 13: Circumscribed circles of the triangles obtained with the function `delaunay`.

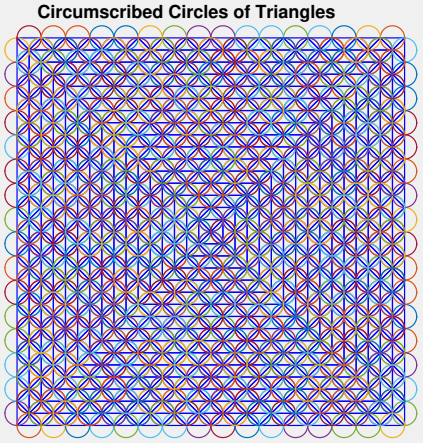

Figure 14: Circumscribed circles of the triangles obtained with the function `delaunay`.

Solution. First we need to compute the center of the inscribed circle and its radius. This task is similar to the one in Problem 2.1. With the same notations a, b, c, S before, the following formulas are used

$$r = \frac{2S}{a+b+c}, \tag{A.1}$$

$$I = \frac{aA + bB + cC}{a+b+c}. \tag{A.2}$$

Here is my MATLAB script for computing the center and radius of the inscribed circle of a triangle in terms of the coordinates of its vertices.

```
function [inscribed_center, inscribed_radius] = inscribed_circle(triangle_coor)
% Compute the center & the radius of the inscribed circle of a triangle whose
% coordinates of its vertices are given
```

```
% Author: Nguyen Quan Ba Hong
% Date: 09/10/2018
% Last Update: 17/10/2018
% Input:
% + triangle_coor: is of size 3x2, gives the coordinates of three vertices
% of a triangle
% Outputs:
% + inscribed_center: the center of the inscribed circle of that triangle
% + inscribed_radius: the radius of the inscribed circle of that triangle

%% Compute the Area of the Triangle
S = polyarea(triangle_coor(:,1), triangle_coor(:,2));

%% Compute the Length of Edges of the Triangle
a = norm(triangle_coor(2,:) - triangle_coor(3,:)); % Length of edge BC
b = norm(triangle_coor(3,:) - triangle_coor(1,:)); % Length of edge CA
c = norm(triangle_coor(1,:) - triangle_coor(2,:)); % Length of edge AB

%% Alternative Way to Compute Area of the Triangle
% S = 1/4*sqrt((a+b+c)*(a+b-c)*(b+c-a)*(c+a-b)); % Heron's formula

%% Compute Inscribed Radius of the Triangle
inscribed_radius = 2*S/(a+b+c);

%% Compute Coordinates of Circumcenter
inscribed_center = 1/(a+b+c)*(a*triangle_coor(1,:) + b*triangle_coor(2,:) ...
    + c*triangle_coor(3,:));
```

Here is my script for the first part:

```
function [triangle_number, max_radius] = find_worst_triangle(mesh)
% Determine the "worst" element of a given mesh, i.e., the element of
% greatest inscribed radius, & that greast value
% Author: Nguyen Quan Ba Hong
% Date: 08/10/2018
% Last Update: 17/10/2018
% Input:
% + mesh: a mesh structure
% Outputs:
% + triangle_number: Index of the "worst" element of the given mesh
% + max_radius: the maximum of the radius of the inscribed circle of the
% triangles of the given mesh
```

```
radius = zeros(size(mesh.triangles,1),1);
for i = 1:size(mesh.triangles,1)
    % Compute Inscribed Radius of the i-th Triangle in the Mesh
    [~, inscribed_radius] = inscribed_circle(mesh.coords(mesh.triangles(i,:),:));
    radius(i) = inscribed_radius;
end
% Determine the maximal radius and the index of the "worst" triangle
[max_radius, triangle_number] = max(radius);
```

and that for the second part:

```
function mesh_out = mesh_by_barycenters(boundary_vertices, mesh_size)
% Build the mesh according to the following strategy:
% 1) For the % initialization of the process: consider the triangulation
% T_0 obtained % from a subdivision of the polygonal boundary of the domain
% of computation. This subdivison would be set of segments of size around
% mesh_size, where mesh_size is the requested size for the elements of the
% mesh.
% 2) Principle: Consider a triangulation T_n given at a step n, determine
% the "worst" triangle (i.e., of greatest inscribed radius), denote G its
% iso-barycenter. The new triangulation T_{n+1} is built by the insert of G
% into the set of points of T_n
% Author: Nguyen Quan Ba Hong
% Date: 08/10/2018
% Last Update: 17/10/2018
% Inputs:
% + boundary_vertices: vector which gives the coordinates of the
% vertices of the polygonal boundary of the computational domain
% + mesh_size: a scalar which specify the required maximal size for the
% elements of the output mesh
% Output:
% + mesh_out: the mesh built according to the described strategy, from a
% subdivision of the boundary of the domain of computation

%% Initialization of the Process: Zeroth Triangulation
coords = [];
% Index the Vertices of Polygonal Boundary, the first vertex is also
% counted again in the last edge, but with the largest numbering
index_boundary_vertices = ones(size(boundary_vertices,1) + 1,1);
for i = 1:size(boundary_vertices,1) % Loop on All Boundary Edges
    % Add Coordinate of the i-th Vertex of Polygonal Boundary to coords
```

```
    coords = [coords; boundary_vertices(i,:)];
    if (i < size(boundary_vertices,1)) % Not the Last Boundary Edge
        % Compute the Length of the i-th Boundary Edge
        length = norm(boundary_vertices(i,:) - boundary_vertices(i+1,:));
        loop = ceil(length/mesh_size); % Number of Points for Partitioning
        for j = 1:loop-1
            % Equally-Spaced Partition of the i-the Boundary Edge
            coords = [coords; (loop-j)/loop*boundary_vertices(i,:) ...
                + j/loop*boundary_vertices(i+1,:)];
        end
    else % The Last Edge in the Polygonal Boundary
        % Compute the Length of the Last Boundary Edge
        length = norm(boundary_vertices(i,:) - boundary_vertices(1,:));
        loop = ceil(length/mesh_size); % Number of Points for Partitioning
        for j = 1:loop-1
            % Equally-Spaced Partition of the Last Boundary Edge
            coords = [coords; (loop-j)/loop*boundary_vertices(i,:) ...
                + j/loop*boundary_vertices(1,:)];
        end
    end
    % Index the (i+1)-th Vertices of the Boundary
    index_boundary_vertices(i+1) = index_boundary_vertices(i) + loop;
end
triangles = delaunay(coords(:,1), coords(:,2));

%% Delete Degenerate Triangles in Delaunay Triangualation
new = []; % Goal: New Triangles Storage without Degenerate Triangles
for i = 1:size(triangles,1)
    flag = 1; % Boolean Variable for Checking Degeneracy of Triangles
    for j = 1:size(boundary_vertices,1) % Loop on All Boundary Edges
        temp = sort(triangles(i,:),2);
        % For the Last Edge of Boundary
        % Condition for Degeneracy of the Triangle on the Last Boundary Edge
        if (temp (1) == 1 && index_boundary_vertices(size(boundary_vertices,1))...
                <= temp(2))
            flag = 0; % Detect Degeneracy
        end
        % For the Non-Last Edge of Boundary
        % Condition for Degeneracy of the Triangle on the Non-Last Boundary Edge
        if ((index_boundary_vertices(j) <= temp(1)) && (temp(3) ...
                <= index_boundary_vertices(j+1)))
            flag = 0; % Detect Degeneracy
```

```
        end
    end
    if (flag == 1) % The Considered Triangle is Not Degenerate
        new = [new; triangles(i,:)]; % Store this Non-Degenerate Triangle
    end
end
triangles = new; % Updated triangles without Degenerate Elements

%% Modify the Triangulation by Adding Iso-Barycenter
while 1
    mesh_out = mesh_new('',coords,triangles); % Update Mesh Structure
    % Find the Worst Element and Its Radius in the Current Mesh
    [triangle_number, max_radius] = find_worst_triangle(mesh_out);
    % Stopping Criteria
    if (max_radius <= mesh_size)
        break % the Current Mesh Fulfills the Requirements
    end
    % Add Iso-barycenter of the Worst Triangle to the Mesh
    coords = [coords; 1/3*sum(coords(triangles(triangle_number,:),:))];
    % Update New Three Triangles Generated by Inserting the Iso-Barycenter
    triangles = [triangles; triangles(triangle_number,[1 2]), size(coords,1);
        triangles(triangle_number,[2 3]), size(coords,1);
        triangles(triangle_number,[3 1]), size(coords,1)];
    triangles(triangle_number,:) = []; % Delete the Worst Triangle
end

%% Create Final Mesh
mesh_out = mesh_new('',coords,triangles);
```

This completes our solution. □

References

[1] https://perso.univ-rennes1.fr/eric.darrigrand-lacarrieu/Teaching/M2RFEorganisation.html