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

Nguyen Quan Ba Hong*

November 9, 2018

Abstract

This context includes the materials given in the course *Finite Element Method* 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).

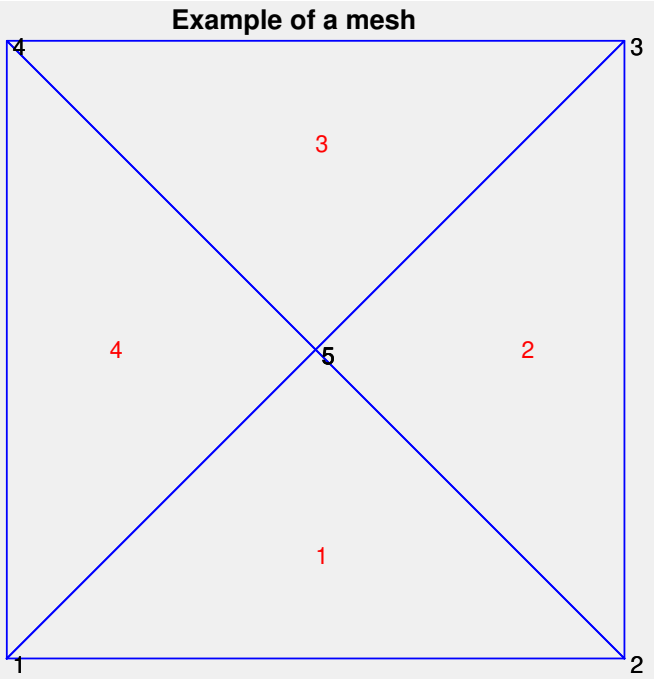

Figure 1: Mesh of the unit square with 4 triangles $\mathbb{P}_1$.

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 has been already stored in the table `edges`. In this case, we skip it and consider the next edge of the triangle, 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.

For illustrations, running the script `build_edge_connectivity` (see Sec. 5.2) in the main script 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.

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

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

The new `coords` for $\mathbb{P}_2$ mesh structure 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. Next, we 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. Hence, we first reuse all the triangles of the $\mathbb{P}_1$ mesh structure as the first three columns of `triangles` for the $\mathbb{P}_2$ mesh structure. Then, we loop on all triangles of the $\mathbb{P}_1$ mesh structure. For each triangle, we make a second loop on all edges of the $\mathbb{P}_1$ mesh structure in order to compare its three edges with all edges. If there is a coincidence, we store the midpoint of that edge in the position 4, 5, or 6, followed by the local numbering convention.

Finally, we generate a $\mathbb{P}_2$ mesh structure by using the tables `coords` and `triangles` just created. See Sect. 5.3 for details.

Running the following commands lines in the main script

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

gives us the following result

[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.

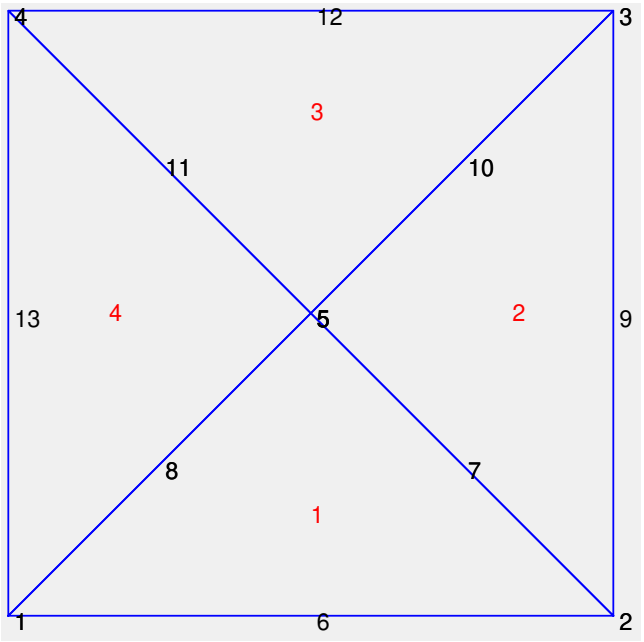

Figure 2: A $\mathbb{P}_2$ mesh structure.

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,

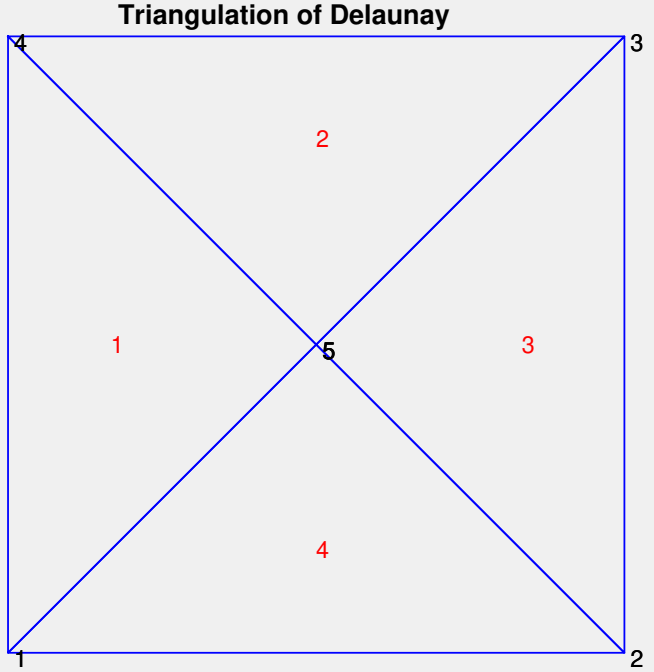

Figure 3: The triangulation of Delaunay has different orders in the numbering, Cf. Fig. 1.

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. See Sect. 5.4 for details.

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

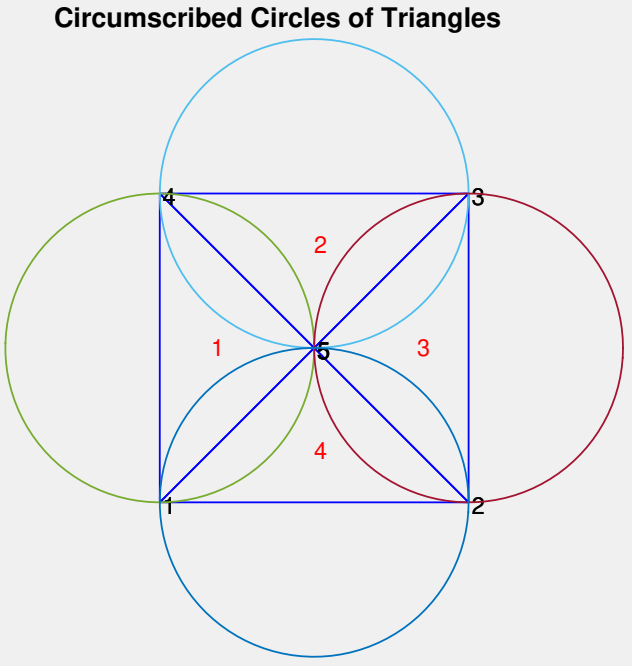

Figure 4: Circumscribed circles of the triangles obtained with the function `delaunay`.

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

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. To do the first task, we first reuse the tables `edges` and `edges_triangles` in Problem 1.1. Then we loop on all edges[3] of the mesh given to compute their lengths. Store all these lengths in a table, say `length_edges`, using the MATLAB built-in routine `max` yields both the maximal value among these lengths and its corresponding index in the table `length_edges`.

[3] If we loop of all triangles of the mesh given instead, all interior edges will be considered twice, so are their lengths

Using this index and the table `edges_triangles`, we obtain the worst triangle as the first[4] triangle containing the longest edge.

For the second task, the first step is to construct the zeroth triangulation $\mathcal{T}_0$. It suffices to construct a table `coords` for it (a table `triangles` then follows by applying the Delaunay triangulation on the table `coords` constructed). The table `coords` are initialized by reusing all the boundary vertices inputted. Then we loop on all the boundary edges of this polygonal domain[5]. For each boundary edge, we compute its length and then perform an equally-spaced partition on it such that the length of each subinterval is less than $\frac{h}{2}$, where h is the required mesh size.

More explicitly, we denote this polygonal domain as $A_1A_2\dots A_n$, where A_i, $i = 1, \dots, n$ are the boundary vertices inputted. Consider the i^{th} boundary edge A_iA_{i+1}, with the convention $A_{n+1} := A_1$ (this is also the reason why the last boundary edge should be treated cautiously), a desired equally-spaced partition is given by

$$\left(1 - \frac{j}{n_i}\right) A_i + \frac{j}{n_i} A_{i+1}, \;\; j = 1, \dots, n_i - 1, \;\text{ where } n_i := 2 \left\lceil \frac{|A_iA_{i+1}|}{h} \right\rceil . \tag{3.1}$$

After obtaining the table `coords`, we perform the Delaunay triangulation to obtain the table `triangles` for $\mathcal{T}_0$. We emphasize here that the Delaunay triangulation considers the nodes in `coords` as being in general position. As a consequence, several degenerate triangles are generated. To handle this trouble, we have to delete all these degenerate triangles. If degenerate elements are ignored, a lot of meaningless computations will be conducted which leads to waste memory and time. Additionally, the quality (see the next section) of a mesh containing degenerate elements will equal ∞ and thus also meaningless.

We now modify the triangulation by the strategy described above. A `while` loop is used until the length of the longest edge in the current mesh is less than or equal to h (stopping criteria). For each loop, we find the worst triangle in the current mesh, and then add its iso-barycenter to the table `coords`. The table `triangles` follows by applying the Delaunay triangulation on the new table `coords`. After generating the new mesh, we delete all degenerate triangles in the current mesh. See Sect. 5.7 for details.

Running the following commands in the main script for $h = 3, 2, 1, 0.5$

```
boundary_vertices = [0,0;3,1;4,4;2,7;-2,8;-4,6;-5,5;-4,1];
mesh_size = 3;
mesh_out = mesh_by_barycenters(boundary_vertices, mesh_size);
% Plot
figure
mesh_plot(mesh_out, 0);
```

yields the following figures

[4] If the longest edge is an interior edge, there are exactly two triangles containing it.

[5] The last boundary edge should be treated with caution.

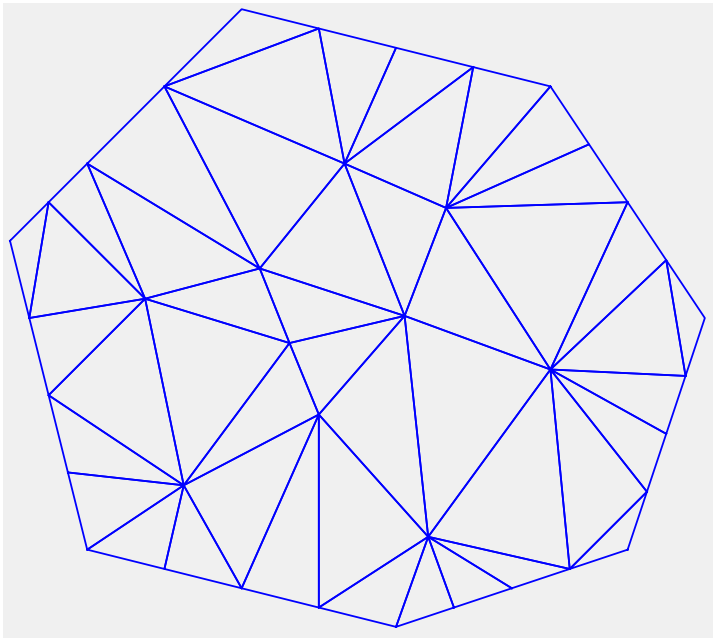

Figure 5: Mesh by barycenters with $h = 3$.

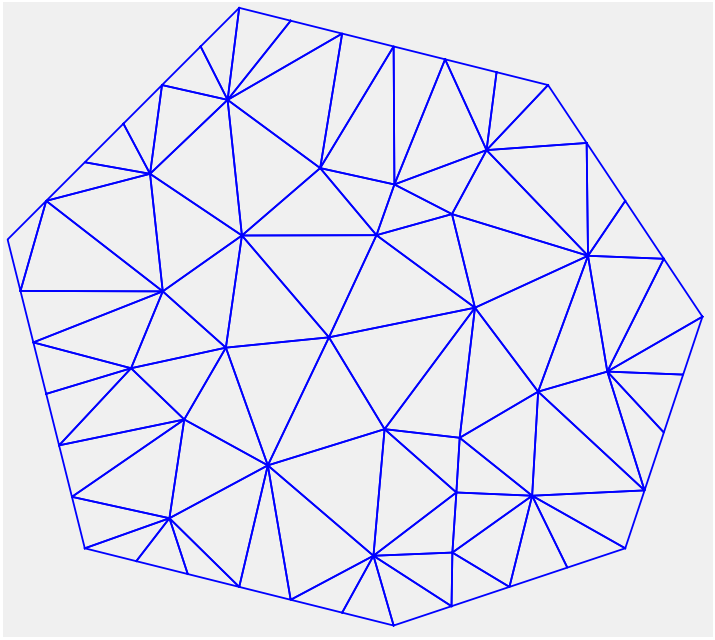

Figure 6: Mesh by barycenters with $h = 2$.

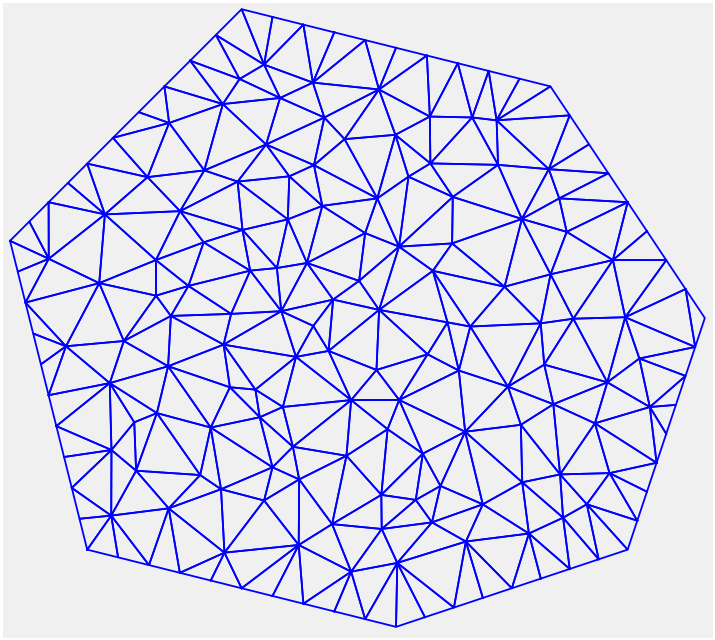

Figure 7: Mesh by barycenters with $h = 1$.

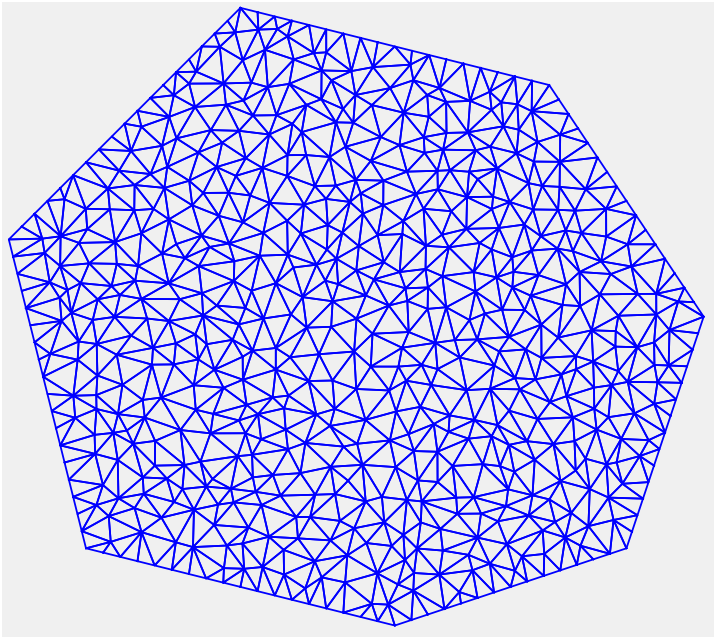

Figure 8: Mesh by barycenters with $h = 0.5$.

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. The first task is straightforward: we compute the radii of the circumscribed and the inscribed circles. The roundness of the triangle whose vertices are inputted is the ratio of the former and the later.

To do the second task, we loop on all triangles in the mesh given. For each triangle, we compute its roundness. After all, the quality of the mesh given is the maximal value of these values. 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. First of all, we need a separate script to find all the nodes on the boundary of a polygonal domain, see Sect. 5.12 for details. We will store all these boundary nodes in the table `coords` for the new mesh. Then we loop on all triangles of the mesh given. For each triangle, we store its iso-barycenter as the new row of the table `coords`. After this, we perform the Delaunay triangulation, generate a new mesh, and delete all its degenerates triangles as above.

Running the following commands in the main script

```
mesh_out = mesh_optimize(mesh_out);
figure(3)
mesh_plot(mesh_out,0);
```

yields the following figures.

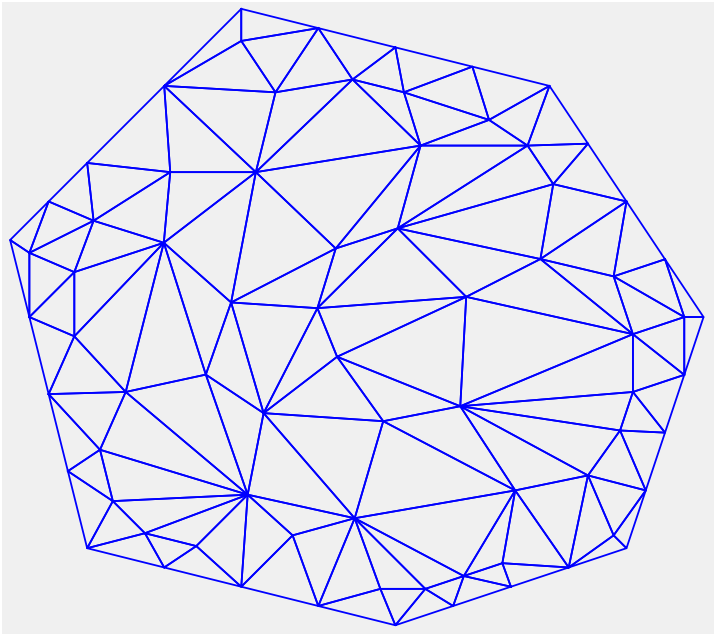

Figure 9: Optimized mesh with $h = 3$.

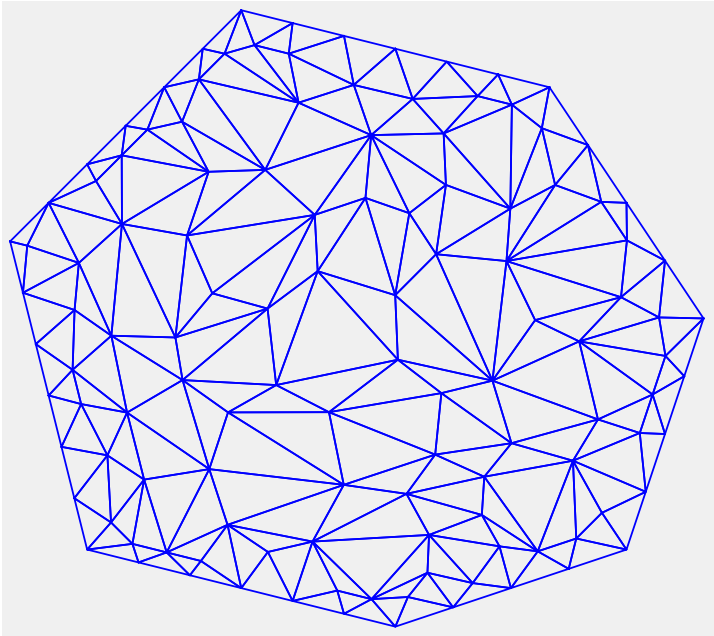

Figure 10: Optimized mesh with $h = 2$.

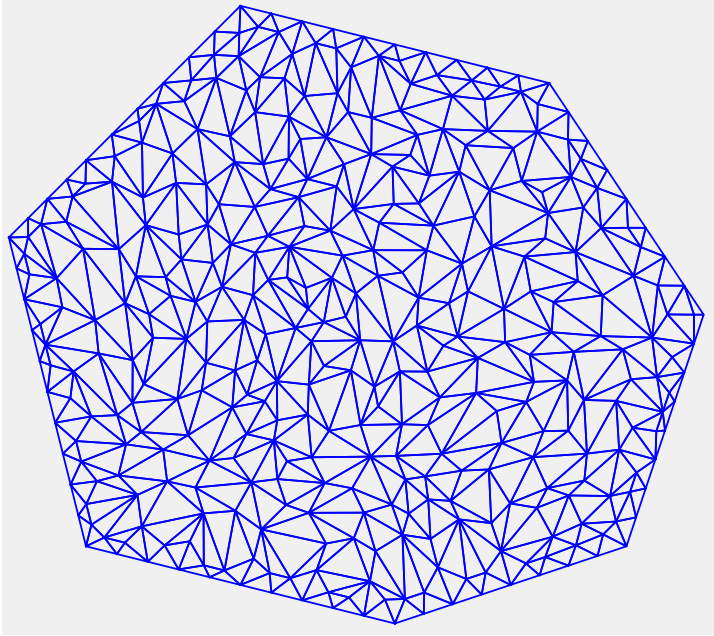

Figure 11: Optimized mesh with $h = 1$.

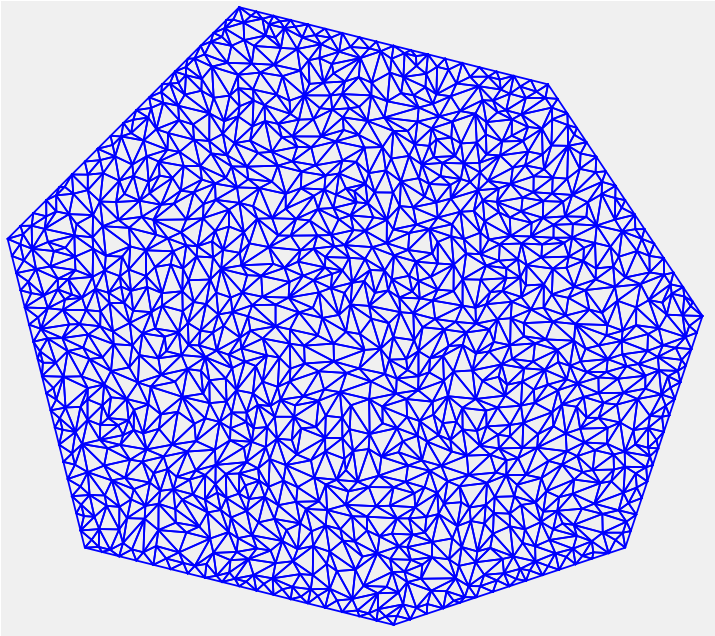

Figure 12: Optimized mesh with $h = 0.5$.

This completes our solution. □

5 Appendices: MATLAB Scripts

5.1 Mesh Constructor

Using the structure of data `struct`, the following MATLAB script generates a mesh structure from two arrays `coords` and `triangles`.

```
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);
```

5.2 Build Edge Connectivity

The following MATLAB script builds the tables `edges` and `edges_triangles` for a mesh, see Sec. 1 for their definitions.

```
function [edges, edges_triangles] = build_edge_connectivity(mesh)
% Build the tables edges & edges_triangles of a mesh given
% Author: Nguyen Quan Ba Hong
% Date: 06/10/2018
% Last Update: 16/10/2018
% Input:
% + mesh: a P_1 mesh structure
% 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 numbered K

%% Build the Table edges
edges = []; % Initialize
for K = 1:size(mesh.triangles,1) % Loop on All Triangles of the Mesh Given
    temp = sort([mesh.triangles(K,[1,2]);
        mesh.triangles(K,[2,3]);
        mesh.triangles(K,[1,3])],2); % Read the 3 Edges of the K-th Triangle
    flag = ones(3,1); % flag = 0: Already Contained; flag = 1: Not Contained Yet
    for i = 1:3 % Loop on 3 Edges of the K-th Triangle
        for j = 1:size(edges,1)
            % Check if the edge considered is already in table edges or not
            if (edges(j,:) == temp(i,:)) % This Edge was Contained in edges
                flag(i) = 0; % Ignore this Edge
            end
        end
        if (flag(i) == 1)
            edges = [edges; temp(i,:)]; % Store the Desired Edge
        end
    end
end

%% Build Cell-Array edges_triangles
for i = 1:size(edges,1) % Loop on All the Edges
    edges_triangles{i} = []; % Initialize
    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]; % First Edge
        elseif (edges(i,:) == sort(mesh.triangles(j,[2,3])))
            edges_triangles{i} = [edges_triangles{i}; j,2]; % Second Edge
        elseif (edges(i,:) == sort(mesh.triangles(j,[1,3])))
            edges_triangles{i} = [edges_triangles{i}; j,3]; % Third Edge
        end
    end
end
```

5.3 Modify $\mathbb{P}_1$ into $\mathbb{P}_2$ Mesh Structure

The following MATLAB script is used to build a $\mathbb{P}_2$ mesh structure from a given $\mathbb{P}_1$ mesh structure.

```
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

%% Create the Table edges
[edges, ~] = build_edge_connectivity(mesh_P1);

%% Construct the Nodes coords of mesh_P2
coords = mesh_P1.coords; % Reuse the Nodes of mesh_P1
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

%% Construct Triangles of mesh_P2
triangles = zeros(size(mesh_P1.triangles,1),6);
triangles(:,[1,2,3]) = mesh_P1.triangles; % Reuse the Triangles of mesh_P1
for K = 1:size(mesh_P1.triangles,1) % Loop on All Triangles of mesh_P1
    for i = 1:size(edges,1) % Loop on All Edges of mesh_P1
```

```
        if (sort(mesh_P1.triangles(K,[1,2])) == edges(i,:)) % First Edge
            triangles(K,4) = size(mesh_P1.coords,1) + i; % First Midpoint
        elseif (sort(mesh_P1.triangles(K,[2,3])) == edges(i,:)) % Second Edge
            triangles(K,5) = size(mesh_P1.coords,1) + i; % Second Midpoint
        elseif (sort(mesh_P1.triangles(K,[1,3])) == edges(i,:)) % Third Edge
            triangles(K,6) = size(mesh_P1.coords,1) + i; % Third Midpoint
        end
    end
end

%% Create a P2-Structured Mesh
mesh_P2 = mesh_new('Mesh_P2', coords, triangles);
```

5.4 Circumscribed Circle of a Triangle Version 1

The following MATLAB script is used to compute the center of the circumscribed circle and its radius for a triangle given through the coordinates of its vertices.

```
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 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 the Area of the Triangle
% S = 1/4*sqrt((a+b+c)*(a+b-c)*(b+c-a)*(c+a-b)); % Heron's Formula
```

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

%% Compute the 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);
```

5.5 Circumscribed Circle of a Triangle Version 2

The following MATLAB script is used to compute the center of the circumscribed circle and its radius for a triangle given through the coordinates of its vertices.

```
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');
```

5.6 Find the Worst Triangle

The following MATLAB script is used to determine the "worst" element of a given mesh.

```
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

%% Create 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);  % Initialize
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);
```

5.7 Mesh by Barycenters

The following MATLAB is used to build the mesh according to the strategy described in Sec. 3, from a subdivision of the boundary of the domain of computation.

```
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/2, 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
i = i+1; % For the Last Boundary Edge
% 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
```

5.8 Roundness of a Triangle

The following MATLAB script is used to compute the roundness of a triangle.

```
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;
```

5.9 Quality of a Mesh

The following MATLAB script is used to compute the quality of a mesh, defined as the largest roundness of the elements.

```
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); % Initialize
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);
```

5.10 Mesh Optimization

The following MATLAB script is used to optimize a mesh.

```
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
boundarynodes = boundary_nodes(mesh_in); % Find All the Boundary Nodes
coords = boundarynodes; % Initialize

%% 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);
```

5.11 Plot of a Mesh

The following MATLAB script is used to plot a mesh structure ($\mathbb{P}_1$ or $\mathbb{P}_2$).

```
function mesh_plot(mesh_s,show_numbers)
% plot a mesh
% Author:G.Vial
% Date: 08/09/2010
% Last update: 14/09/10
%
% usage: mesh_plot(mesh_s,show_numbers)
%
% input -
%   mesh_s: mesh structure
%   show_numbers: boolean(true if numbers are displayed)
%
```

```
triangles = getfield(mesh_s,'triangles');
coords = getfield(mesh_s,'coords');
nb_triangles = size(triangles,1);
name = getfield(mesh_s,'name');
%
trimesh(triangles(:,1:3),coords(:,1),coords(:,2),'Color','b');
%
if (show_numbers)
   % offset for numbering
   offset = max(max(coords))/100;
   for k = 1:nb_triangles
      triangle = triangles(k,:);
      coord = coords(triangle,:);
      % display global numbers
      for v = 1:size(triangle,2)
         text(coord(v,1)+offset,coord(v,2)-offset,num2str(triangle(v)));
      end
      % display triangle number
      G = mean(coord(1:3,:));
      text(G(1),G(2),num2str(k),'Color','red');
   end
end
%
% axis and title
axis equal
axis off
title(name,'FontWeight','bold')
xlabel('x')
ylabel('y')
```

5.12 Boundary Nodes of a Mesh

The following MATLAB script is used to compute the coordinates of the nodes on the boundary of a given mesh.

```
function boundary_nodes = boundary_nodes(mesh)
% Produce a table containing the coordinates of the nodes on the
% boundary of a given mesh
% Author: Nguyen Quan Ba Hong
% Date: 30/10/2018
% Last Update: 30/10/2018
% Input:
```

```
% + mesh: a mesh structure
% Output:
% + boundary_nodes: a table of size (the number of nodes on boundary)x2
% containing the coordinates of the nodes on the boundary of the mesh given

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

%% Take All Boundary Nodes of the Mesh Given
boundary_nodes = []; % Initialize
for i = 1:size(mesh.coords,1) % Loop on All Nodes of the Given Mesh
    % Check if the Node is Interior or Not
    for j = 1:size(edges,1) % Loop on All Edges of the Given Mesh
        % Check if the Edge Contains the i-th Node or Not
        if (edges(j,1) == i || edges(j,2) == i)
            if (size(edges_triangles{j},1) == 1) % This is a Boundary Edge
               boundary_nodes = [boundary_nodes; mesh.coords(i,:)];
               break
            end
        end
    end
end
```

5.13 Delete All Degenerate Triangles of a Mesh

The following MATLAB script is used to delete all the "degenerate triangles" in a given mesh. A "degenerate triangle" is defined as a triangle whose the radius of its corresponding inscribed circle satisfies $r_K \leq tol$, where tol is some given tolerance.

```
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-2
% 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);
```

5.14 Inscribed Circle of a Triangle Version 1

The following MATLAB is used to compute the center of the inscribed circle and its radius, for a triangle given through 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,:));
```

5.15 Inscribed Circle of a Triangle Version 2

The following MATLAB is used to compute the center of the inscribed circle and its radius, for a triangle given through the coordinates of its vertices.

```
function [incenter,inradius] = incircle(triangle_coor)
% Compute the inscribed 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: [incenter,inradius] = incircle(triangle_coor)
%
% input -
%   triangle_coor : 3x2 array containing the coordinates of the vertices of the triangle.
% output -
%   incenter : center of the inscribed circle
%   inradius : radius of the inscribed circle
%
A = triangle_coor(1,:);
B = triangle_coor(2,:);
C = triangle_coor(3,:);
AB = sqrt((B-A)*(B-A)'); %'
AC = sqrt((C-A)*(C-A)'); %'
BC = sqrt((C-B)*(C-B)'); %'
G = (1./(AB+AC+BC))*(BC*A+AC*B+AB*C);
incenter = G;
AG = sqrt((G-A)*(G-A)'); %'
vABAC = cross([B-A,0],[C-A,0]);
ABAC = sqrt(vABAC*vABAC'); %'
angleCAG = 0.5*asin(ABAC / (AB*AC));
inradius = AG * sin(angleCAG);
```

5.16 Local Numbering

The following MATLAB script is used to indicate the local numbering convention, which may be useful for other purposes.

```
function a = local_numbering(b)
% Define the local numbering of the reference triangles P_1 & P_2
% Author: Nguyen Quan Ba Hong
% Date: 16/10/2018
% Last Update: 16/10/2018
% Input:
% + b: a 1x1 or 1x2 matrix
% Output:
% + a: If b is 1x1: b = 1 (edge 1) return a = [1,2], b = 2 (edge 2) return
% a = [2,3], b = 3 (edge 3) return a = [1,3]. If b is 1x2: b = [1,2] return
% a = 1, b = [2,3] return a = 2, b = [1,3] return a = 3

if (b == 1)
    a = [1,2];
elseif (b == 2)
    a = [2,3];
elseif (b == 3)
    a = [1,3];
elseif (b == [1,2])
    a = 1;
elseif (b == [2,3])
    a = 2;
elseif (b == [1,3])
        a = 3;
else
    error('Invalid input');
end
```

5.17 Mesh Refinement

The following MATLAB script is used to refine a given $\mathbb{P}_1$ mesh by adding the midpoints of all the edges of that mesh.

```
function mesh_out = refine_mesh(mesh)
% Refine a mesh by adding the midpoints of all edges of the mesh given
% Author: Nguyen Quan Ba Hong
% Date: 06/10/2018
```

```
% Last Update: 08/10/2018
% Input:
% + mesh: a P1 mesh structure
% Output:
% + mesh_out: a refined mesh

%% Create the Table edges for the Mesh Given
[edges, ~] = build_edge_connectivity(mesh);

%% Construct coords of mesh_P2
coords = mesh.coords; % Reuse all the Nodes of the Mesh Given
for i = 1:size(edges,1) % Loop on All Edges of the Mesh Given
    % Add the Midpoint of i-th Edge to the New Table coords
    coords = [coords; (coords(edges(i,1),:) + coords(edges(i,2),:))/2];
end

%% Construct the Table triangles for the New Mesh
triangles = []; % Initialize
for i = 1:size(mesh.triangles,1) % Loop on All Triangles of the Mesh Given
    for j = 1:size(edges,1) % Loop on All Edges of the Mesh Given
        if (edges(j,:) == sort([mesh.triangles(i,1), mesh.triangles(i,2)]))
            midpoint_edge1 = j + size(mesh.coords,1);
        end
        if (edges(j,:) == sort([mesh.triangles(i,2), mesh.triangles(i,3)]))
            midpoint_edge2 = j + size(mesh.coords,1);
        end
        if (edges(j,:) == sort([mesh.triangles(i,1), mesh.triangles(i,3)]))
            midpoint_edge3 = j + size(mesh.coords,1);
        end
    end
    triangles = [triangles; mesh.triangles(i,1),midpoint_edge1,midpoint_edge3;
        mesh.triangles(i,2),midpoint_edge1,midpoint_edge2;
        mesh.triangles(i,3),midpoint_edge2,midpoint_edge3;
        midpoint_edge1,midpoint_edge2,midpoint_edge3];
end

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

Refining successively the mesh above gives us the following figures.

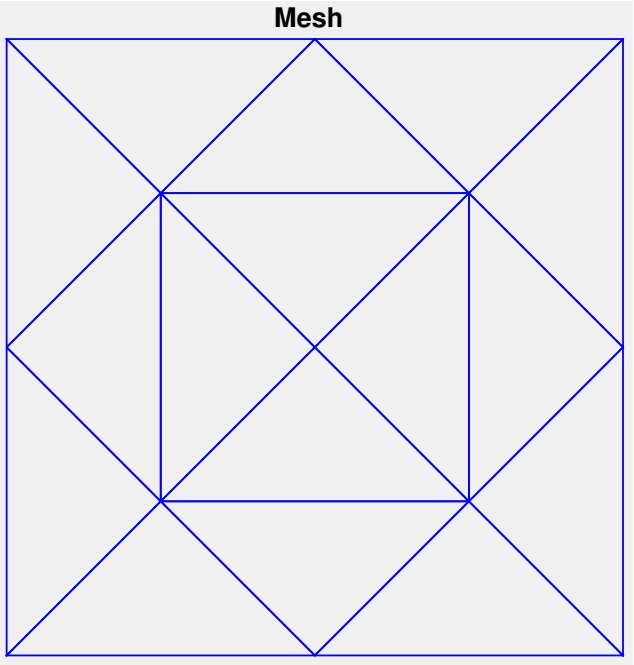

Figure 13: Refine mesh once without numbering.

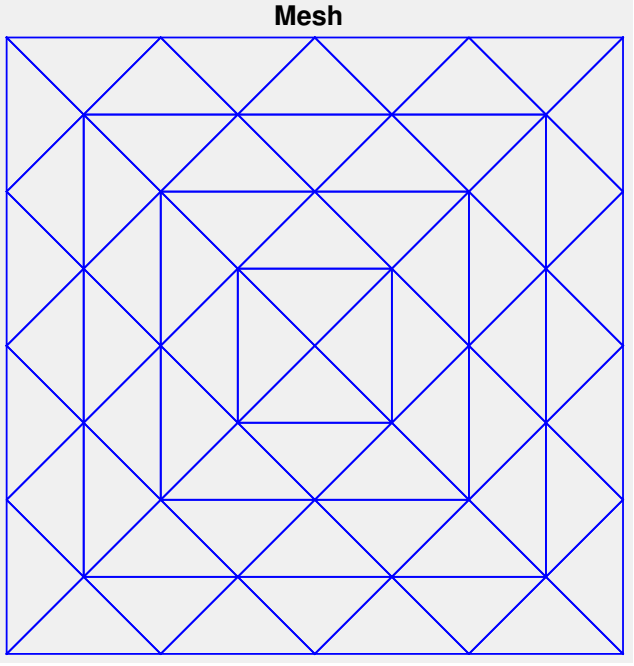

Figure 14: Refine mesh twice without numbering.

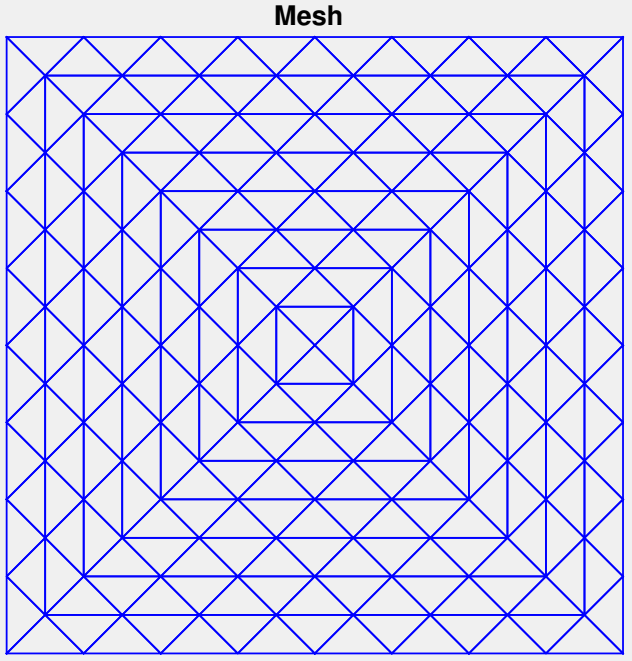

Figure 15: Refine mesh three times without numbering.

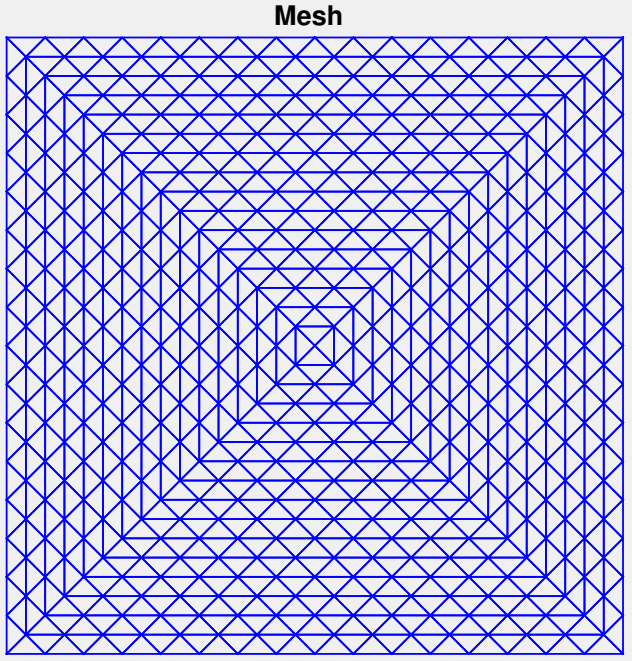

Figure 16: Refine mesh four times without numbering.

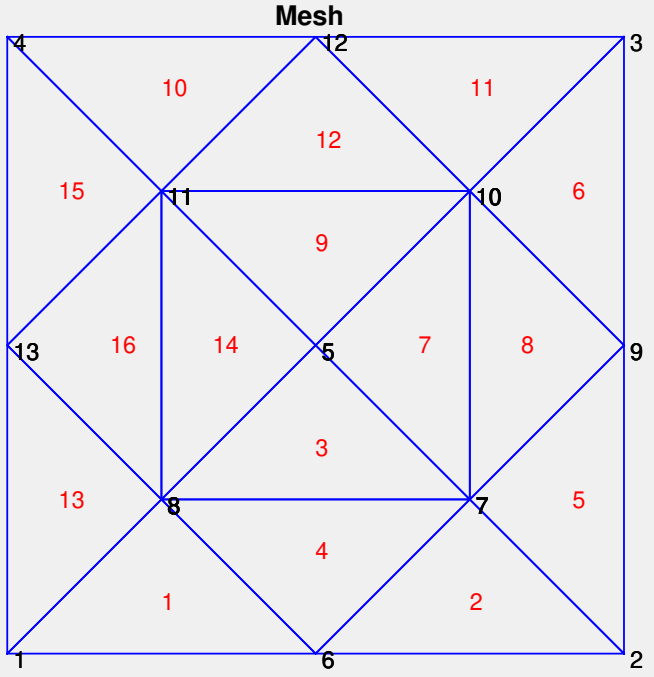

Figure 17: Refine mesh once with numbering.

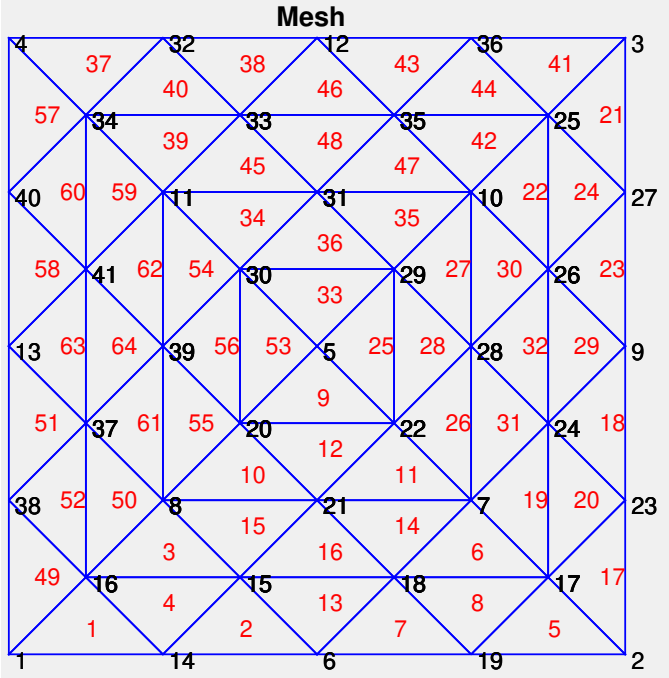

Figure 18: Refine mesh twice with numbering.

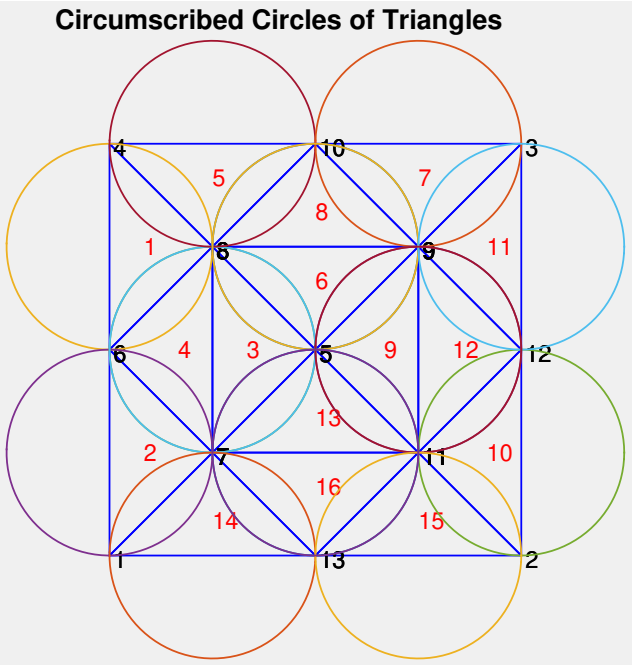

Figure 19: Circumscribed circles of all triangles in Fig. 13.

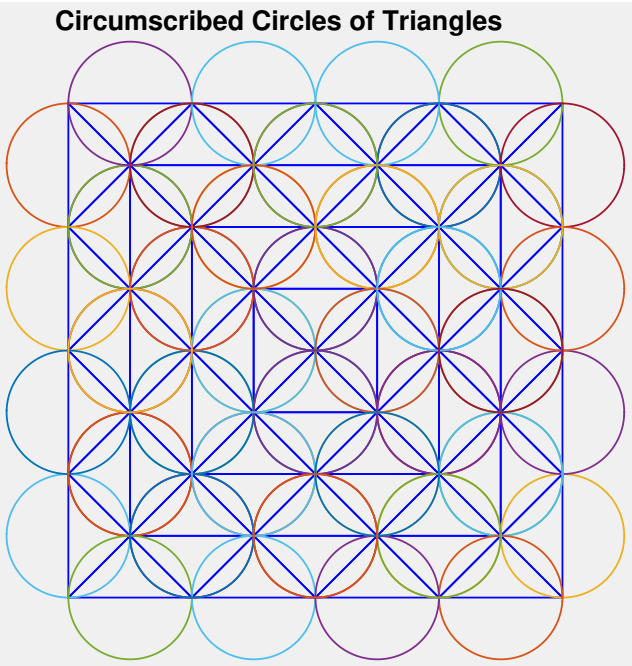

Figure 20: Circumscribed circles of all triangles in Fig. 14.

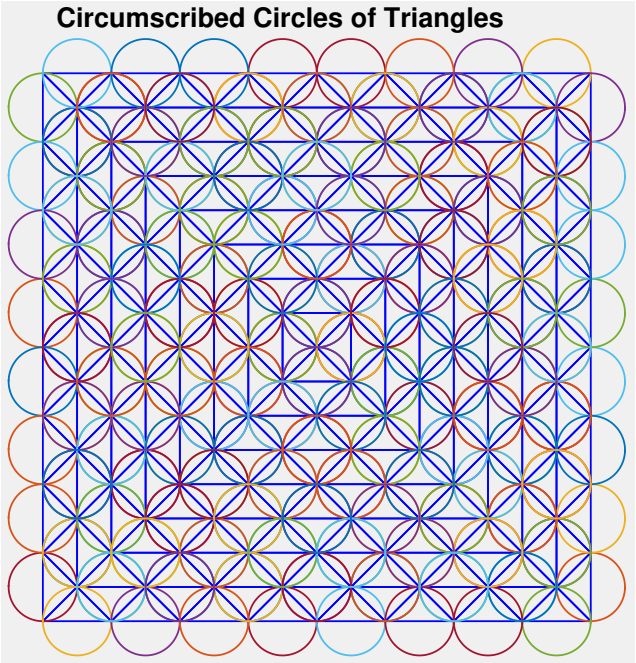

Figure 21: Circumscribed circles of all triangles in Fig. 15.

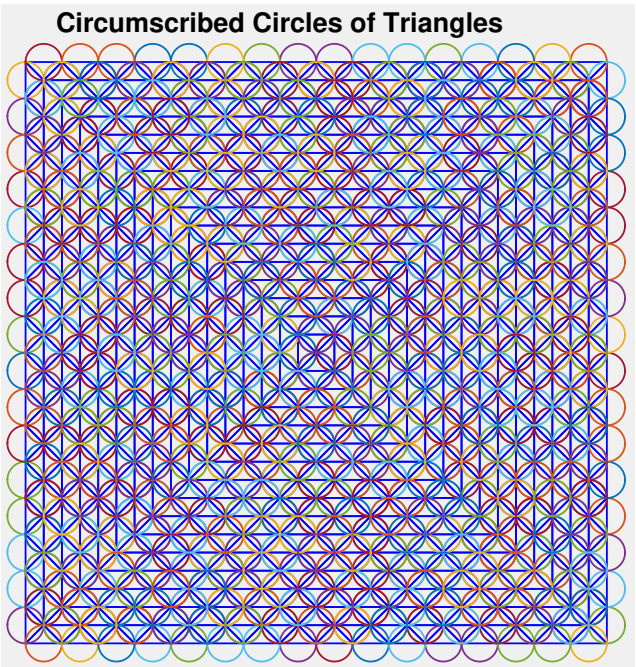

Figure 22: Circumscribed circles of all triangles in Fig. 16.

References

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