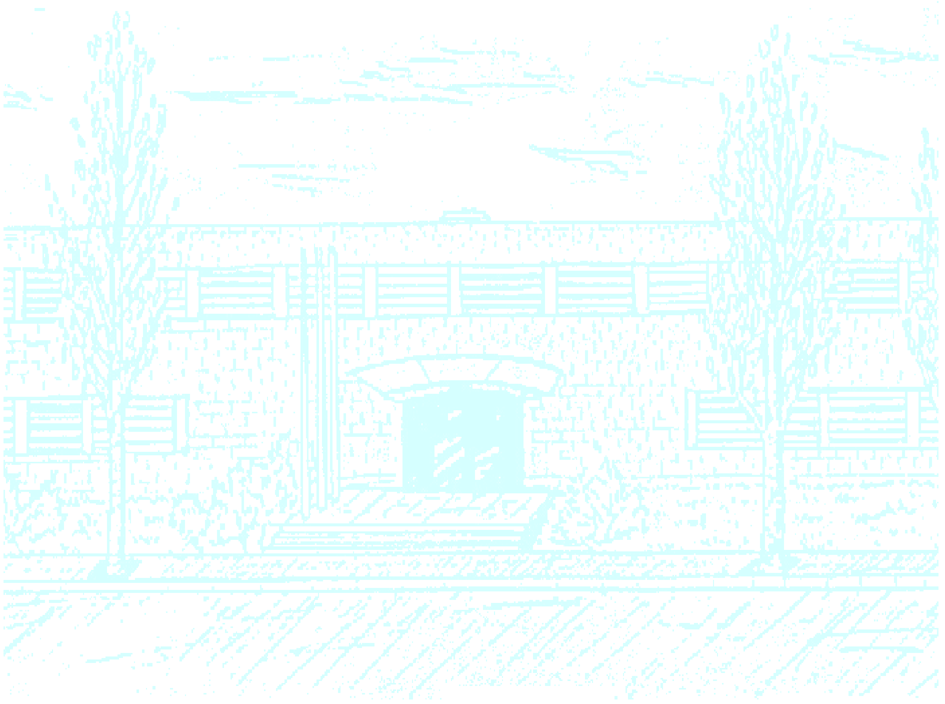
Title: Smoothing and untangling of meshes on parameterized surfaces

Author: Abel Gargallo Peiró

Advisors: Xevi Roca Navarro and Josep Sarrate Ramos

Department: MA3

Academic year: 2010-2011



Universitat Politècnica de Catalunya

Facultat de Matemàtiques i Estadística

Projecte Final de Màster

Smoothing and untangling of meshes on parameterized surfaces

Author:
Abel Gargallo Peiró

Advisors:
Xevi Roca Navarro
Josep Sarrate Ramos

Departament de Matemàtica Aplicada 3

Barcelona
January 31, 2011

Acknowledgments

This master thesis could not have been done without the help of all the people that has been surrounding me during this last year.

First, I would like to specially thank Josep Sarrate, my Technological Project and Mater thesis advisor, for letting me the opportunity to work with him. In a very smooth way he introduced me my first research project and never forced my path, letting me choose every step that I wanted to take. Was everything your strategy to little by little induce me to the dark side that is the doctoral thesis? Even if it was or not, thank you for everything.

Second, I want to also thank really much Xevi Roca, also advisor of this Master thesis, for never closing the door of his office to me. Even when I was not his official student, he offered me all what he knew and all the time that he was able to lend me. I really appreciate your attention, your patience and your advice.

Third, I would like to thank all the rest of the Lacàn members. I must underline the help that has always offered to me Eloi Ruiz, who has been forced to solve an innumerable amount of questions, and whose meshing work has been inspiring me to learn more and more in order to be able to have a PhD thesis in the next years that can be compared to his. I would like to thank all the rest of Lacàn one by one, but you are two much, and you know, I would like my thesis to be larger than my acknowledgements.

I can not forget about the friends that I have gathered during my degree. Without your support in long afternoons working on projects and exams where everyone tried to give his best, becoming a mathematician would have been even more difficult. Special thanks to Carlos, that despite being a pure algebra mathematician has always supported me.

I also want to thank my family for their support. Without their help and all what my parents and my grandparents have offered to me all my life I am sure that I would not be here. Every advice, every small thing that they taught me has driven me to be who I am, and this is something I will never forget. I would like to specially thank the backing that my brother has always given to me.

Finally, I want to thank Maite for all her support and understanding. Without her advice and her smile all the work would have been harder.

Abstract

Surface meshes composed by triangular and quadrilateral elements are one of the most used techniques to approximate and represent continuous surfaces. Numerical simulation requires to use surface meshes composed by non-folded and positive oriented elements with a shape close to a regular polygon. To ensure these requirements, there are several techniques to both untangle (unfold) and smooth (morph to regular shapes) surface meshes. However, there is not a technique that both untangles and smooths meshes taking into account the surface parameterization. Note that in numerical simulation surface meshes are usually generated from a computer aided design (CAD) model, and therefore the surface parameterization is available. Thus, the main contribution of this work is to present a new technique to both untangle and smooth meshes on parameterized surfaces. To this end, we extend to parameterized surfaces the existent untangling and smoothing techniques for planar meshes and non-parameterized surfaces. Specifically, we propose an extension of any planar element quality measure to elements on parameterized surfaces. This quality measure is the basis of the proposed smoothing and untangling procedure. In addition, for quadrilateral elements we propose a new objective function that requires less computational effort than the standard one. The examples show that the proposed objective function also provides the same or even better element quality. We also modify the minimization process of the standard untangling and smoothing techniques. Successful untangling and smoothing techniques use a modified objective function to ensure the convergence of the minimization process (robustness). This modification is controlled by a fixed scalar parameter that is determined heuristically. However, for meshes with non-uniform distributions of the element size this approach do not ensure convergence. To overcome this issue, we have proposed two alternative modifications of the objective function. The resulting minimization process is more robust and converges for all the tested surface meshes. Finally, we present the techniques used to improve the performance of the final implementation of the method.

Keywords: mesh untangling; mesh smoothing; mesh quality; optimization; quadrilateral meshes; triangular meshes; surface meshes; parameterized surfaces; mesh generation; finite element method.

Contents

Chapter 1

Introduction

The Finite Element Method is nowadays one of the most used techniques in applied sciences and engineering. The application of the method requires a previous discretization of the geometry. The accuracy of the FEM depends on the quality of this discretization. On the one hand, this discretization has to capture properly the geometry. On the other hand, this discretization has to be composed by well-shaped elements that satisfy certain geometrical requirements. Therefore, the use of FEM in industrial applications is then slowed down by the need of a generation of a good mesh.

It is well known that the precision of the numerical solution obtained by the FEM depends on the size and the shape of the elements of the mesh. Several quality measures have been defined in order to quantify the deviation of the shape of an element respect to an "ideal" shape. For instance, [Field] presents a comparative analysis of several quality measures for triangles and tetrahedrons.

It is important to point out that meshing algorithms are hierarchic procedures. Thus, in order to mesh a 3D object we first have to mesh its 2D boundary. Consequently, the 2D mesh will also require a previous 1D discretization. Therefore, the quality of a 3D mesh is directly affected by the quality of the boundary discretization (surface mesh). Thus, it is of the major importance to generate a high quality surface mesh.

Several techniques have been developed in order to improve the shape of the elements of a given mesh. They can be classified in two main categories. The first one is composed by those procedures that modify the topology (the connectivity) of the mesh. That is, some elements are removed or modified in order to generate new ones that have better quality. The second one, is composed by those techniques that modify the geometry of the mesh (the location of the nodes), without changing its topology, in order to obtain a better configuration of the mesh. That is, they "smooth" the location of the nodes.

A wide range of smoothing algorithms have been developed during the last decades (see for instance [Herrman] and [Giuliani], among others). It is important to point out that these methods are based on geometrical and/or numerical reasoning. In general, these algorithms are fast from the computational point of view. However, they are not robust, in the sense that they can move nodes outside of the domain in complex geometries (for instance on convex corners). In addition, these algorithms are not designed to maximize a given quality measure.

[Knupp 01] introduced a family of quality measures placed within an algebraic framework that have been intensively used during the last decade. Later, [Knupp 03b] proposed a smoothing method based on an optimization of these measures. In fact, this optimization procedure is transformed into a continuous minimization problem.

These optimization algorithms are more robust than the previous ones. However, they are still not able to untangle inverted elements. [Escobar 03] introduced a modification of the measures developed by Knupp in which this lack was covered. The optimization of the new objective function was able to simultaneously untangle and smooth a mesh, saving time and effort in order to obtain the final mesh.

Later, [Escobar 06] extended this algorithms to non-planar triangular meshes. This new method is based on the projection of the surface mesh into a projection plane. This projection plane is determined by an additional optimization problem that increases the computational cost of the global smooth algorithm. In this plane the local mesh is smoothed and then is taken back to the surface, approximating the new location on the surface mesh.

The aim of this work is to develop a simultaneous smoothing-untangling procedure for parametrized surfaces. To this end we will extend the definition of the quality metrics for planar meshes to parametrized surface meshes. Then, we will develop a minimization approach on the parametric space that will allow smoothing and untangling the surface mesh. To this end, first we will increase the robustness of the standard untangling techniques. Then, we will focus on the improvement of the computational efficiency of the developed approach. Finally, several examples will be presented in order to illustrate the capabilities of the proposed method.

Chapter 2

Algebraic quality measure

2.1 Basics on the quality of an element

The *quality metric* of an element (triangle or quadrilateral in 2D problems, tetrahedron or hexahedron in 3D) is a scalar function such that measures a given geometric property of the analysed element. It is usually a function defined on the vertices of the element.

To fix notation, we denote the physical space dimension by n ($n = 2$ for 2D problems and $n = 3$ for 3D problems). Moreover, let m be the number of vertices of the element[1], and $\mathbf{x}_k \in \mathbb{R}^n$ the coordinates of those vertices. Taking into account this notation, the *quality metric* is defined as the following scalar function

$$\begin{array}{rcl} q : \mathbb{R}^n \times \overset{(m)}{\cdots} \times \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ (\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}) & \longmapsto & q(\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}). \end{array}$$

According to [Knupp 01], any quality metric should hold the following properties:

- q is dimension-free.
- q is going to be referenced to an ideal element that describes the desired shape of the element.
- For all $\mathbf{x}_k$ in the domain, $q(\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}) \in [0, 1]$. That is:

  $$q : \mathbb{R}^n \times \overset{(m)}{\cdots} \times \mathbb{R}^n \longrightarrow [0, 1].$$

  Note that q will only be 1 if the element achieves its ideal configuration, and it will only be 0 if the element is degenerated[2].

[1]For instance $m = 3$ vertices for triangles and $m = 4$ for tetrahedrons, or $m = 4$ for quadrilaterals and $m = 8$ for hexahedra.

[2]A 2D element will be considered degenerated if it has area 0. Equivalently, it will be degenerated in 3D if its volume is 0.

- q is invariable under translations or rotations of the element.
- q does not depend on the numbering of the nodes of the element.

In this work we will use a *shape quality metric* introduced in [Knupp 03]. The aim of this metric is to detect the distortions in the shape of the element, letting apart its size.

2.2 Planar quality measure for triangular elements

2.2.1 Jacobian matrix and reference and ideal elements

Let t_R be a *reference element* delimited by vertices $\mathbf{u}_0 = (0,0)$, $\mathbf{u}_1(1,0)$ and $\mathbf{u}_2 = (0,1)$ in a logical space. We want to find an affine mapping $\mathbf{f}$ that maps this reference element onto a given triangle, t, in the physical space, defined by nodes $\mathbf{x}_0 = (x_0, y_0)$, $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$, see Figure 2.1.

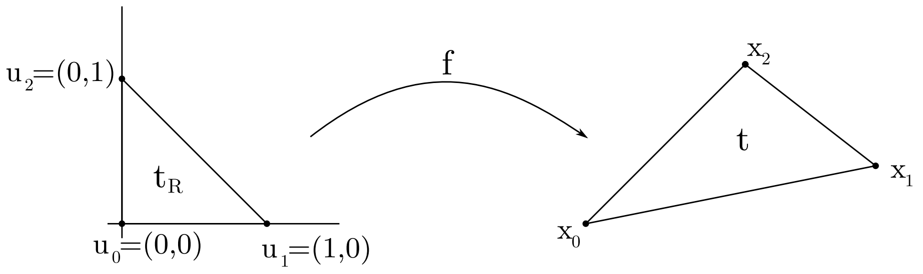


Figure 2.1: Affine mapping between the reference and the physical element.

Let ξ_k, $k = 0,1,2$, be the barycentric coordinates of any point inside the reference triangle. Recall that the barycentric coordinates verify that $0 \le \xi_k \le 1$ for $k = 0,1,2$ and $\sum_{k=0}^{2} \xi_k = 1$. Then, taking into account these properties the affine mapping can be written as

$$\mathbf{f}(\xi_1, \xi_2) \equiv \mathbf{f}(\xi_0, \xi_1, \xi_2) = \sum_{k=0}^{2} \xi_k \mathbf{x}_k \overset{\xi_0+\xi_1+\xi_2=1}{=} (1 - \xi_1 - \xi_2)\mathbf{x}_0 + \xi_1 \mathbf{x}_1 + \xi_2 \mathbf{x}_2. \quad (2.1)$$

This mapping can also be written in matrix form as

$$\begin{array}{rcl} \mathbf{f} : t_R & \longrightarrow & t \\ \mathbf{u} & \longmapsto & \mathbf{x} = \mathbf{A_0}\mathbf{u} + \mathbf{x}_0, \end{array} \quad (2.2)$$

where

$$\mathbf{A_0} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix},$$

and $\mathbf{x} = (x, y)^T$, $\mathbf{u} = (\xi_1, \xi_2)^T$.

Note that this application maps $\mathbf{u}_0$, $\mathbf{u}_1$ and $\mathbf{u}_2$ in the logical space onto $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$ in the physical space. The vector $\mathbf{x}_0$ controls the translation of the element, and the matrix $\mathbf{A_0}$ controls its area, shape and orientation. Matrix $\mathbf{A_0}$ is called the *Jacobian matrix*, because it is indeed the Jacobian of the affine mapping with respect to $\{\xi_k\}_{k=0,1,2}$.

We denote by *ideal element*, t_I, the element that represents the desired shape to achieve. Therefore, to measure the deviation from the ideal triangle, t_I, of any triangle t in the physical space, we want to find an affine mapping, $\mathbf{f_S}$, such that $\mathbf{f_S} : t_I \longrightarrow t$ (see Figure 2.2).

According to Equation (2.2) it is really simple to go from the reference triangle in the logical coordinates to any other triangle in the physical space. Therefore, we use this idea to determine $\mathbf{f_S}$ by the composition of two functions.

To this end, we first define the affine mapping between the reference element, t_R, and the ideal one, t_I. If $\tilde{\mathbf{x}}_k$ are the coordinates of the ideal element, this affine mapping can be written as (see Figure 2.2)

$$\begin{array}{rcl} \mathbf{f_W} : \; t_R & \longrightarrow & t_I \\ \mathbf{u} & \longmapsto & \mathbf{x} = \mathbf{W}\mathbf{u} + \tilde{\mathbf{x}}_0, \end{array}$$

where

$$\mathbf{W} = \begin{pmatrix} \tilde{x}_1 - \tilde{x}_0 & \tilde{x}_2 - \tilde{x}_0 \\ \tilde{y}_1 - \tilde{y}_0 & \tilde{y}_2 - \tilde{y}_0 \end{pmatrix}.$$

Similarly we define the affine mapping $\mathbf{f_A}$, that maps the reference element t_R to the physical triangle t, with coordinates $\mathbf{x}_k$, $k = 0, 1, 2$, as (see Figure 2.2)

$$\begin{array}{rcl} \mathbf{f_A} : \; t_R & \longrightarrow & t \\ \mathbf{u} & \longmapsto & \mathbf{x} = \mathbf{A}\mathbf{u} + \mathbf{x}_0, \end{array}$$

where

$$\mathbf{A} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}, \tag{2.3}$$

Thus, the desired affine mapping $\mathbf{f_S}$ that maps the ideal element onto the physical triangle can be defined as:

$$\mathbf{f_S} = \mathbf{f_A} \circ \mathbf{f_W^{-1}} : \begin{array}{ccccc} t_I & \xrightarrow{\mathbf{f_W^{-1}}} & t_R & \xrightarrow{\mathbf{f_A}} & t \\ \tilde{\mathbf{x}} & \longmapsto & \mathbf{u} = \mathbf{f_W^{-1}}(\tilde{\mathbf{x}}) & \longmapsto & \mathbf{x} = \mathbf{f_A}(\mathbf{u}). \end{array} \tag{2.4}$$

Therefore, the analytical expression of affine mapping $\mathbf{f_S}$ is

$$\mathbf{f_S}(\tilde{\mathbf{x}}) \;=\; \mathbf{f_A}(\mathbf{f_W^{-1}}(\tilde{\mathbf{x}})) = \mathbf{A}\mathbf{f_W^{-1}}(\tilde{\mathbf{x}}) + \mathbf{x}_0 = \mathbf{A}(\mathbf{W^{-1}}\tilde{\mathbf{x}} + \tilde{\mathbf{v}}) + \mathbf{x}_0 = \mathbf{A}\mathbf{W^{-1}}\tilde{\mathbf{x}} + \mathbf{v},$$

for a given translation vector $\mathbf{v} = \mathbf{A}\tilde{\mathbf{v}} + \mathbf{x}_0$.

The Jacobian matrix of this application

$$\mathbf{S} = \mathbf{A}\mathbf{W}^{-1} \tag{2.5}$$

is called *shape matrix* in references [Knupp 03] and [Escobar 03].

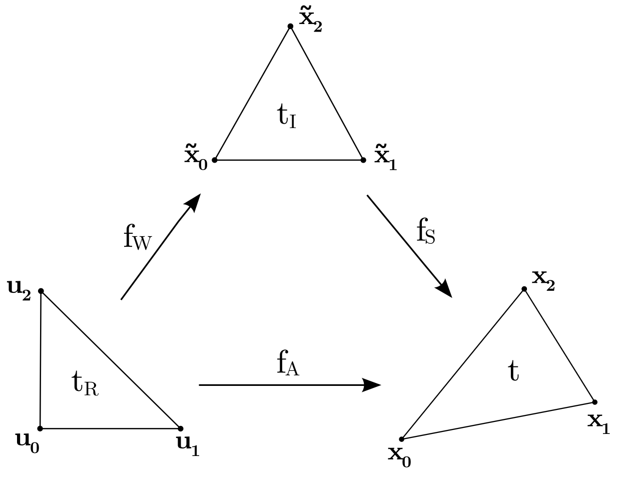


Figure 2.2: Affine mappings for triangular elements.

[Knupp 01] proves that the affine mapping $\mathbf{f_S}$ defined in (2.4) does not depend on the node that has been chosen as translation vector or on the numbering of the nodes.

Note that we can select different ideal elements in order to generate elements with different geometric properties. The following examples will illustrate this idea.

- For isotropic triangular meshes we choose as ideal element the equilateral triangle. Taking $\tilde{\mathbf{x}}_0 = (0,0)^T$, $\tilde{\mathbf{x}}_1 = (1,0)^T$ and $\tilde{\mathbf{x}}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})^T$ as the coordinates of the equilateral triangle, the resulting Jacobian matrix is

$$\mathbf{W} = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{pmatrix}. \tag{2.6}$$

- For quadrilateral meshes, each element, delimited by the nodes $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, is divided into four triangles $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2\}$, $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_3\}$, $\{\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_3\}$ and $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ (see Figure 2.3). The quality of the quadrilateral is a weighting of the quality of this four triangles (see references [Knupp 03] and [Knupp 03b]). The ideal quadrilateral element is the square. Thus, if we subdivide it this way into four triangles, we get four triangles that are indeed rectangle and isosceles. Therefore, in this case the triangle that represents the geometric property that we want to achieve is not the equilateral but the rectangle isosceles. Then, if we choose $\tilde{\mathbf{x}}_0 = (0,0)$, $\tilde{\mathbf{x}}_1 = (1,0)$ and $\tilde{\mathbf{x}}_2 = (0,1)$, the Jacobian matrix becomes the identity matrix

$$\mathbf{W} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{2.7}$$

because the logical coordinates are indeed those of the ideal triangle.

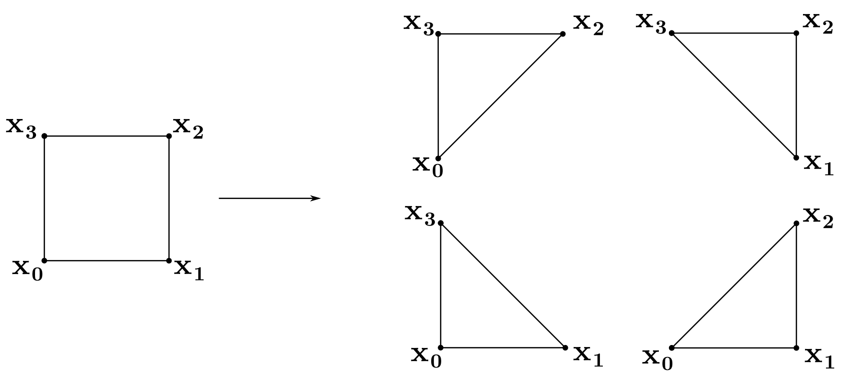


Figure 2.3: Division of a square in order to compute its quality.

2.2.2 Planar quality measure for triangles

The shape matrix $\mathbf{S}$ defined in (2.5) contains information about how much we have "distorted" the ideal element to become the physical one. In this section we will analyse a quality metric for planar triangular meshes based on $\mathbf{S}$. Later, this planar measure will be extended to surface problems.

In this work we will use the *shape metric*, introduced by [Knupp 01],

$$\eta(\mathbf{S}) = \frac{|\mathbf{S}|^2}{n\sigma(\mathbf{S})^{2/n}} \overset{n=2}{=} \frac{|\mathbf{S}|^2}{2|\sigma(\mathbf{S})|}. \tag{2.8}$$

where $\sigma(\mathbf{S}) = \det(\mathbf{S})$ is the determinant of $\mathbf{S}$, and $|\mathbf{S}| = \sqrt{(\mathbf{S},\mathbf{S})} = \sqrt{\mathrm{tr}(\mathbf{S^T S})}$ is its Frobenius norm[3]. The parameter n is the dimension of the space we are working in, and hence, $n = 2$.

The image of this function is the interval $[1,\infty]$, achieving ∞ only when the physical element is degenerated, and 1 when it becomes the ideal triangle. Thus, we define the *quality index* as the inverse of the shape metric (see [Knupp 03])

$$q(\mathbf{S}) = \frac{1}{\eta(\mathbf{S})} = \frac{2|\sigma(\mathbf{S})|}{|\mathbf{S}|^2}, \tag{2.9}$$

[3]We use the notation $(\mathbf{A},\mathbf{B}) = \mathrm{tr}(\mathbf{A}^T\mathbf{B})$.

Note that the quality index (2.9) behaves then, as we have previously commented: it reaches a maximum value of 1 for the ideal element, and a minimum value 0 for degenerated elements.

Recall that we want to improve the quality of the mesh by means of a minimizing problem. Since the shape metric takes its minimum value for the ideal element and goes to infinity for the degenerated case, we use it to design our approach to the mesh optimization procedure. Hence, we also name this metric as *objective function* of the analysed element (see [Escobar 03] and [Escobar 06]), since we use it as the basic element to define the function that we finally minimize.

To illustrate the behaviour of the quality index $q(\mathbf{S})$, defined in (2.9), and the objective function $\eta(\mathbf{S})$, defined in (2.8), we introduce the following example. Consider a triangular element with two fixed nodes, $\mathbf{x}_0 = (0, 0.5)$ and $\mathbf{x}_1 = (0, -0.5)$, and a third node $\mathbf{x}_2(x) = (x, 0)$ that we move in the x-axis, see Figure 2.4.

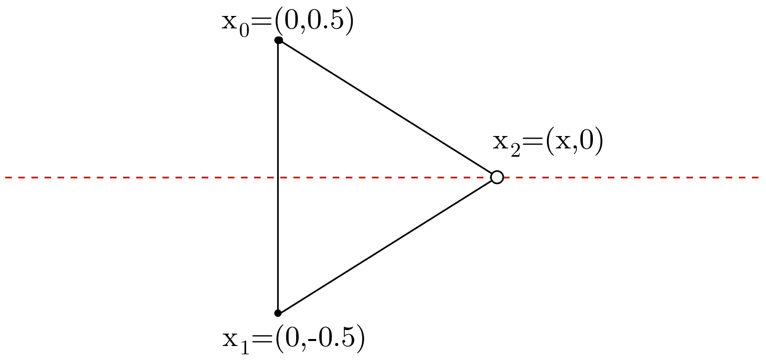


Figure 2.4: Triangle moving the node $\mathbf{x}_2$.

In Figures 2.5(a) and 2.5(b) the quality index and the objective function are displayed when node $\mathbf{x}_2$ moves from $x = -5$ to $x = 5$. Note that we have selected the equilateral triangle as ideal element.

Figure 2.5(a) shows that there is just a point with quality zero. This is the point at which the element achieves its degenerated configuration. Note that the quality index tends to zero when $x \to \pm\infty$, because the limit $\pm\infty$ can also be considered as a degenerated position.

Figure 2.5(a) also shows that the quality index has two maximums, achieved for $x = \pm\sqrt{3}/2$. In these two points, the value of the measure is 1, because the equilateral (ideal) configuration is achieved. However, when we have the triangle in a mesh, it inherits an order in the nodes from the connectivity matrix. Lets consider that the analysed triangle inherits the nodal order $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2\}$. Then, the only valid position to place the nodes is in the right side domain ($x = \sqrt{3}/2$). The other maximum ($x = -\sqrt{3}/2$) leads to an inverted configuration that is not acceptable in a mesh.

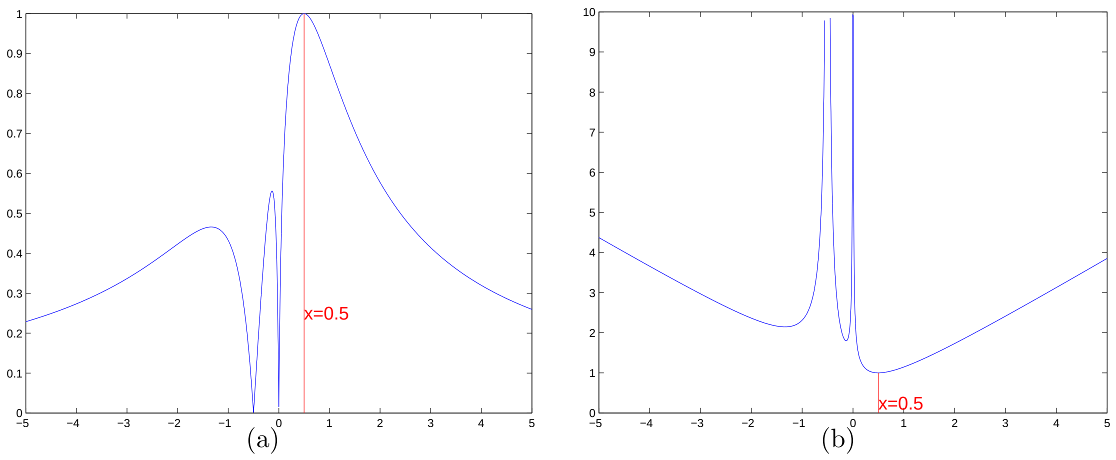


Figure 2.5: (a) Quality index $q(\mathbf{S}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2(x)))$ and (b) objective function $\eta(\mathbf{S}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2(x)))$, moving $\mathbf{x}_0(x)$ for $x \in [-5, 5]$.

Figure 2.5(b) shows that the optimal locations for node $\mathbf{x}_2$ are at the two minimums. Moreover, an asymptote has appeared at $x = 0$, due to the achievement of the degenerated configuration ($\sigma(\mathbf{S}) = 0$).

Note that function (2.8) is not able to untangle elements because it only measures the shape of the elements, but it does not consider their orientation. Thus, it can not distinguish the correct position among the two minima. These difficulties will be overcome on Section 2.6.

2.3 Planar quality measure for quadrilateral elements

As we have already introduced in Section 2.2.1, in order to define the quality of a quadrilateral element, we shall divide it into four triangles defined by each vertex of the quadrilateral and the two adjacent ones (see Figure 2.3). Recall that we anticipated this idea in order to remark that the ideal triangle is no necessarily always the equilateral one. Now that we are strictly working with quadrilaterals, lets formulate a more precise development of the quality of a quadrilateral element.

In Section 2.2, we have revised the construction of a quality measure for triangles that holds the properties required by reference [Knupp 01]. It has been a simple and useful construction based on three main elements: the physical element, and the ideal and reference ones. The physical element was the triangle that is being analysed, the ideal element was the equilateral one, and the reference element was the isosceles triangle (with vertices $(0, 0), (1, 0), (0, 1)$), which lets us define affine mappings between the other triangles in a simple way.

Taking advantage of the simplicity of the construction for triangular elements, we use this theory in order to define a quality index for quadrilateral elements. To this end, we also need an ideal quadrilateral element in order to compare it with the physical quadrilateral, and thus, determine its quality. The *ideal quadrilateral* is the square because it represents the shape that we would like for all the quadrilaterals of a mesh.

However, there is no simple way to directly develop an analogous quality theory for quadrilaterals. Thus, the quality measure for a quadrilateral is defined through the decomposition of the quadrilateral into four triangles (recall Figure 2.3). The quality measure is defined then by a weighting of the sum of the quality functions of the four triangles.

Since we have commented that the ideal quadrilateral is the square, the ideal triangle is becomes then isosceles rectangle triangle, due to the fact that if one decomposes the way we have exposed a square into triangles, the resulting triangles are isosceles rectangle. Thus, lets proceed defining the quality measure for a quadrilateral with nodes $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$.

If we denote by $\eta(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \equiv \eta(\mathbf{S}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k))$ the shape metric of a triangular element defined by nodes $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k$, the extended definition for a quadrilateral $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ is presented in Equation (2.10) (see [Knupp 03] and [Knupp 03b]):

$$\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) + \eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_3) + \eta(\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_3) + \eta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)}{4}. \quad (2.10)$$

Note that since Equation (2.10) achieves its minimum when all the triangles are isosceles rectangle and will tend to infinite when any of the subtriangles becomes degenerated, since the shape metric of such degenerated triangle tends to infinity.

Similarly to Equation (2.9), we define the *quality index of a quadrilateral* as the inverse of the shape metric

$$q(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{1}{\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)}. \quad (2.11)$$

Note that the quality function $q(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ behaves as introduced on Section 2.1, holding all the points that the definition of quality function requires. Specifically, it will take value 1 for the ideal quadrilateral, since the shape metric of all the subtriangles will take value 1. Moreover, it will take value 0 when any of the the subtriangles is degenerated (and so it is the quadrilateral), since the shape metric of such triangle will tend to infinity, and thus, $q(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \to 0$.

In order to illustrate the behaviour of the quality index $q(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, presented in (2.11), and the objective function $\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, defined in (2.10), for a given quadrilateral $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, we introduce the following example: consider a quadrilateral element with three fixed nodes, $\mathbf{x}_1 = (0, 0.5)$ and $\mathbf{x}_2 = (-0.5, 0)$ and $\mathbf{x}_3 = (0, -0.5)$, and a fourth node $\mathbf{x}_0(x) = (x, 0)$ that we can move on the x-axis, see Figure 2.6.

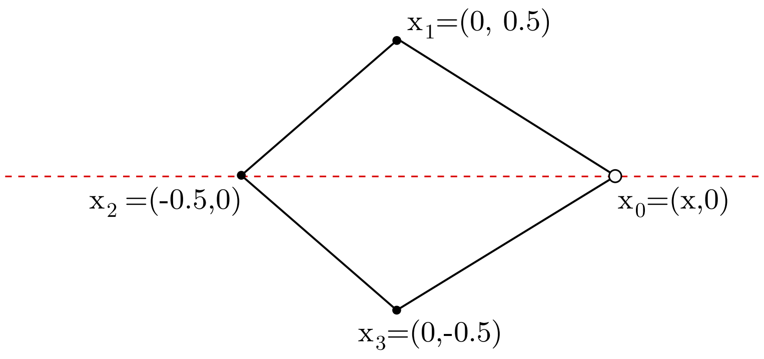


Figure 2.6: Quadrilateral element with a moving the node, $\mathbf{x}_0$.

Figures 2.7(a) and 2.7(b) show the quality index and the objective function when node $\mathbf{x}_0$ moves from $x = -5$ to $x = 5$. Note that we have selected the square as ideal element.

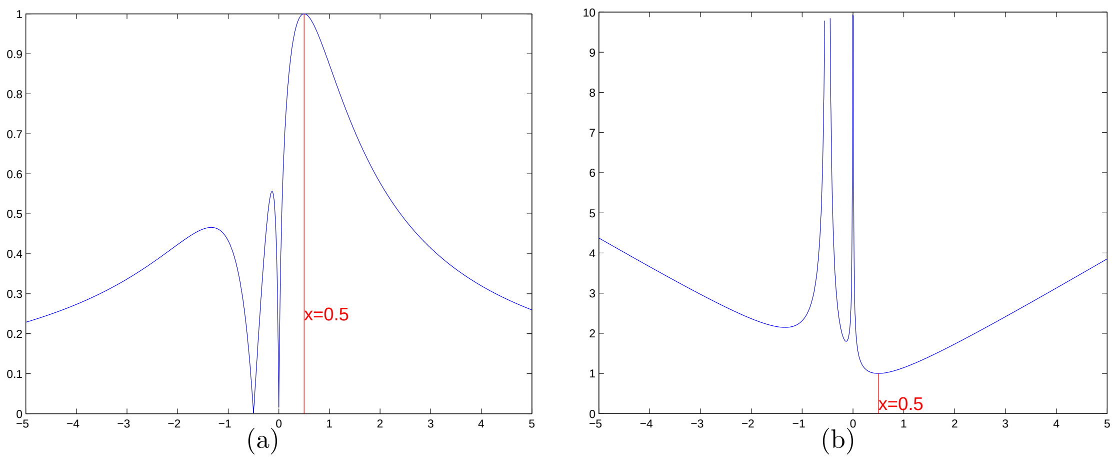


Figure 2.7: (a) Quality index $q(\mathbf{x}_0(x), \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and (b) objective function $\eta(\mathbf{x}_0(x), \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, moving $\mathbf{x}_0(x)$ for $x \in [-5, 5]$.

The first remarkable property comparing these plots to the analogous ones for triangles is the fact that the quadrilaterals have just one global maximum (minimum for the objective function). While in the triangle case we can always have two possible ideal configurations, the decomposition of the quadrilateral element into several triangles allows only one ideal position of the free node. Thus, for triangles the correct position of the free node can only be chosen taking on account the orientation of the triangle, but, for quadrilaterals, the own definition of the shape metric does not let any room for several possible configurations.

However, the number of local maxima/minima increases and hence, it also does increase the difficulty to untangle the quadrilateral if the free node is in a tangled

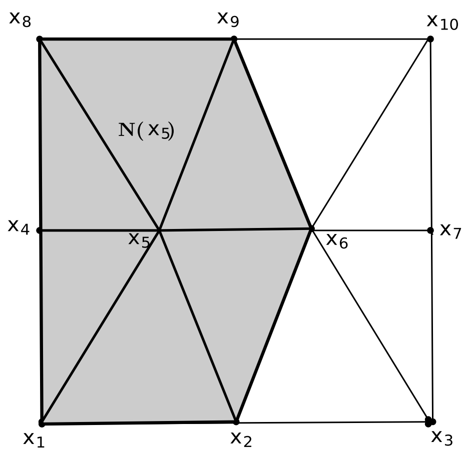


Figure 2.8: Triangular mesh with 2 inner nodes. Marked in grey the local submesh $\mathcal{M}_v$, for $v \equiv \mathbf{x}_5$.

position. In Section 2.6 we will introduce a technique that is able to untangle triangular and quadrilateral meshes by modifying the definition of the triangular objective function (2.8).

Finally, note that it is a bit harder than for triangles to read the behaviour of the quality/objective function in the central domain, since the decomposition of the quadrilateral definition into four triangles amplifies the casuistry (this will be better exemplified in Section 2.5 when working with a whole quadrilateral mesh).

2.4 The objective function for a triangular mesh

Suppose that we have a given mesh $\mathcal{M}$, instead of just an element as we had in Section 2.2.2. In order to improve the quality of all the elements of the mesh (smooth the mesh) we will modify the location of the inner nodes. Thus, all boundary nodes will be fixed. Let $\mathcal{V}$ be the set of inner nodes and let v be a given node $v \in \mathcal{V}$.

Given a node $v \in \mathcal{V}$ we define the *local submesh* associated to it, $\mathcal{M}_v$, as the set of elements that contain node v. Figure 2.8 shows the local mesh associated to the inner node $\mathbf{x}_5$. The coordinates of the nodes of the mesh in Figure 2.8 are:

$$\begin{array}{ll}
\mathbf{x}_1 = (0,0) & \mathbf{x}_2 = (\frac{1}{2},0) \\
\mathbf{x}_3 = (1,0) & \mathbf{x_4} = (0,\frac{1}{2}) \\
\mathbf{x}_5 = (\frac{1}{3},\frac{1}{2}) & \mathbf{x_6} = (\frac{1}{6},\frac{1}{2}) \\
\mathbf{x_7} = (1,\frac{1}{2}) & \mathbf{x_8} = (1,0) \\
\mathbf{x_9} = (1,\frac{1}{2}) & \mathbf{x_{10}} = (1,1)
\end{array}$$

The objective function on node v will be computed as a weighting of the contribution of all the elements that belong to its local mesh.

Let $\mathbf{x} = (x, y)$ be the coordinates of the inner node v, and let $\mathcal{M}_v$ be the associated local submesh. Assume that $\mathcal{M}_v$ is composed by m elements (triangles in our case).

Let $\mathbf{S}_k$ be the Jacobian matrix of the kth triangle of $\mathcal{M}_v$. Then, according to [Knupp 03b], we write the objective function (or shape metric) on the kth triangle as

$$\eta_k(\mathbf{x}) = \frac{|\mathbf{S}_k(\mathbf{x})|^2}{2\sigma(\mathbf{S}_k(\mathbf{x}))}. \tag{2.12}$$

We define the objective function on node v as the p-norm[4] of the objective function of all the triangles of its local submesh:

$$K_\eta^p(\mathbf{x}) = \left(\sum_{k=1}^{m} (\eta_k)^p (\mathbf{x}) \right)^{1/p}. \tag{2.13}$$

In this context, we define the *feasible region* as the set of points where the free node can be located to get a valid mesh. Concretely, the feasible region is the interior of the polygonal set $\mathcal{H} = \cap_{k=1}^{k=m} H_k$ where H_k are the half-planes defined by $\sigma_k(\mathbf{x}) \geq 0$. We say that a triangle of the mesh is *inverted* if $\sigma(\mathbf{x}) < 0$, and *degenerated* if $\sigma(\mathbf{x}) = 0$.

Figure 2.9 presents the surface and the contour plots of the objective function corresponding to node 5 of the mesh presented in Figure 2.8.

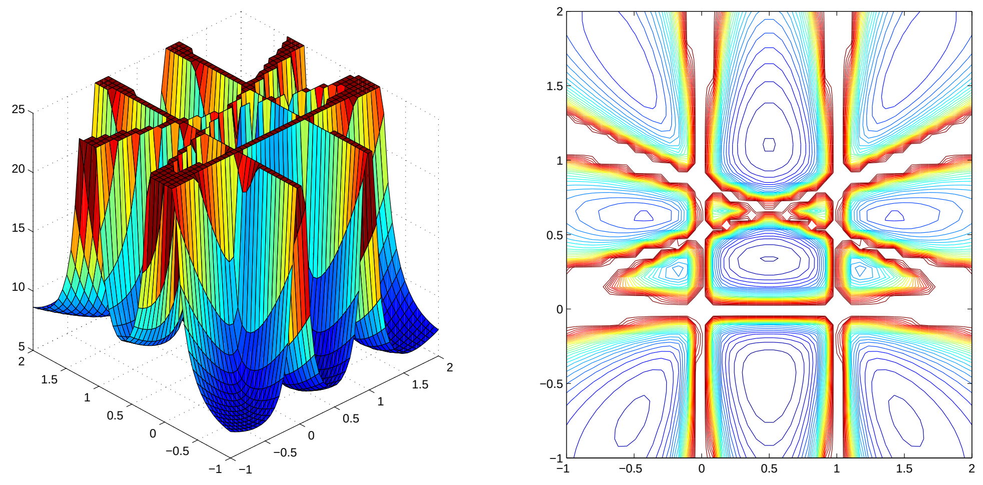


Figure 2.9: Objective function of node 5 of the mesh of Figure 2.8, with a fixed threshold equal to 25.

[4]In this project we will consider by default the 2-norm, p=2.

The objective function (2.13) has several asymptotes at the boundary of the feasible region, where $\sigma = 0$. This avoids the optimization algorithm creating a tangled mesh when it starts from a valid position of node $\mathbf{x}_5$. However, these asymptotes do not allow the optimization algorithm to reach a valid position when it starts from a tangled one ($\mathbf{x}_5$ initially placed in a position that defines a tangled triangle).

Moreover, the objective function also presents local minima outside the feasible region. Then, the optimization algorithm, that is indeed a minimization, will find a minimum that is not the optimal position. Furthermore this local minimum will generate a tangled triangle.

2.5 The objective function for a quadrilateral mesh

Given a quadrilateral mesh $\mathcal{M}$ and a certain local submesh $\mathcal{M}_v$ associated to an inner node v, let $\mathbf{x} = (x, y)$ be the coordinates of the node v. Assuming that $\mathcal{M}_v$ is composed by m quadrilaterals, let η_k be the objective function of the kth quadrilateral that contains node v, see Equation (2.15).

Analogously to the triangular case, we define the objective function on node v as the p-norm of the objective function of all the quadrilaterals of its local submesh:

$$K_\eta^p(\mathbf{x}) = \left(\sum_{k=1}^{m} (\eta_k)^p (\mathbf{x}) \right)^{1/p}. \tag{2.14}$$

However, note that at the same time, if η_k is the objective function for the kth quadrilateral, it is indeed defined by $\eta_k = \eta_k^1 + \eta_k^2 + \eta_k^3 + \eta_k^4$, where $\eta_k^j,\ j = 1, \ldots, 4$ denotes the jth triangle that defines the objective function of the analysed quadrilateral (Equation (2.15)).

Figure 2.10 shows the local mesh associated to the inner node $\mathbf{x}_6$. The coordinates of the nodes of the mesh in Figure 2.10 are:

$$\begin{array}{ll}
\mathbf{x}_1 = (0,0) & \mathbf{x}_2 = (\frac{1}{3},0) \\
\mathbf{x}_3 = (\frac{2}{3},0) & \mathbf{x_4} = (1,0) \\
\mathbf{x}_5 = (0,0.5) & \mathbf{x_6} = (\frac{1}{3},0.5) \\
\mathbf{x_7} = (\frac{2}{3},0.5) & \mathbf{x_8} = (1,0.5) \\
\mathbf{x_9} = (0,1) & \mathbf{x_{10}} = (\frac{1}{3},1) \\
\mathbf{x_{11}} = (\frac{2}{3},1) & \mathbf{x_{12}} = (1,1)
\end{array}$$

Figure 2.11 presents the contour plot of the objective function corresponding to node 6 of the mesh presented in Figure 2.10. Note that for the quadrilateral case we avoid displaying the surface plot, since the number of local minima would not let a proper understanding of the figure.

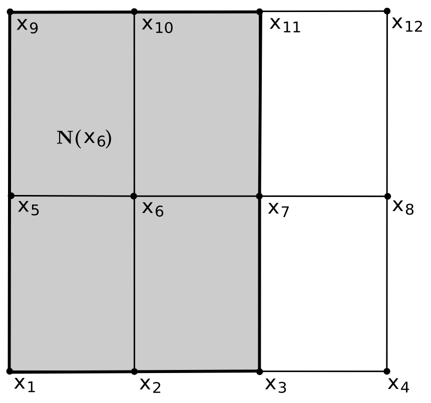


Figure 2.10: Quadrilateral mesh with 2 inner nodes. Marked in grey the local submesh $\mathcal{M}_v$, for $v \equiv \mathbf{x}_5$.

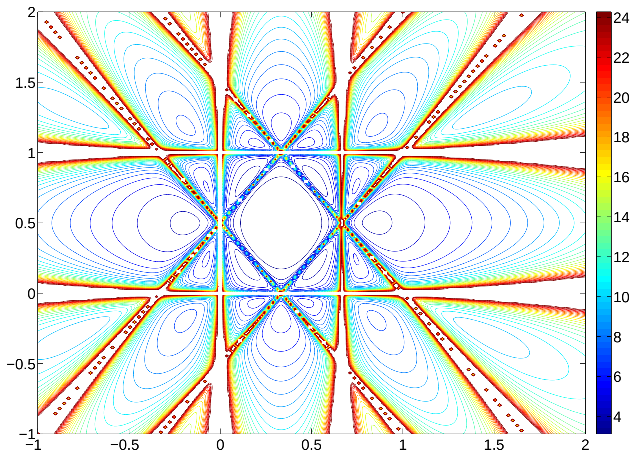


Figure 2.11: Objective function of node 6 of the mesh of Figure 2.10, with a fixed threshold equal to 25.

As it already happened for the triangle case, the objective function (2.14) has several asymptotes at the boundary of the feasible region, where $\sigma = 0$ for any of the triangles that define the objective function of the four quadrilaterals that compose the selected submesh. The same observations that were brought up for triangular meshes still hold. On the one hand, the asymptotes prevent the optimization algorithm of creating a tangled mesh when the optimization starts from a valid position of node $\mathbf{x}_6$. On the other hand they do not allow the optimization algorithm to reach a valid position when it starts from a tangled one. Note that now the objective function presents even more local minima outside the feasible region, due to the increase in the number of underlying triangles needed to define the objective function.

2.5.1 Computational improvement of the standard planar objective function for a free node in a quadrilateral mesh

In this section analyse the definition of objective function that we have brought up for quadrilateral elements. We will find that it can be computationally improved when used in an optimization procedure through a minimization process.

Equation (2.10) defines the objective function of any quadrilateral element as a weighting of the objective functions related to the four underlying triangles, see Figure 2.3.

In an optimization procedure, we loop on the submeshes defined by each inner node and we will improve the location of the inner node maintaining the boundary nodes fixed. Thus, if we place ourselves in one of the quadrilaterals of the submesh and we let $\mathbf{x} \equiv \mathbf{x}_0$ be the free node, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ are constants.

Hence, ,

$$\begin{aligned}\frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3) &= \frac{1}{4}\left(\frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2) + \frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_3) + \frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_2,\mathbf{x}_3) + \overbrace{\frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3)}^{0}\right) \\ &= \frac{1}{4}\left(\frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2) + \frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_3) + \frac{\mathrm{d}\eta}{\mathrm{d}\mathbf{x}}(\mathbf{x},\mathbf{x}_2,\mathbf{x}_3)\right).\end{aligned}$$

Therefore, when computing the derivatives, the fourth triangle does not provide any information. Furthermore, note that in an optimization procedure of node $\mathbf{x}$ the contribution to the quadrilateral objective function of the constant triangle $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ can be directly eliminated, since it is a constant and its elimination will not modify the location of the global minimum.

Thus, we can redefine the objective function as follows:

$$\eta(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3) = \frac{\eta(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2) + \eta(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_3) + \eta(\mathbf{x}_0,\mathbf{x}_2,\mathbf{x}_3)}{3}, \qquad (2.15)$$

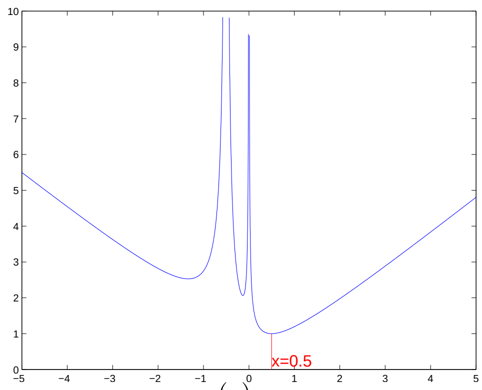
being $\mathbf{x}_0$ the free node.

Note that we cannot take this expression any more as a shape index to define a quality measure, since it does not measure correctly the overall quality of the analysed quadrilateral element. However, note that in an optimization procedure it performs as the standard objective function defined from the four underlying triangles.

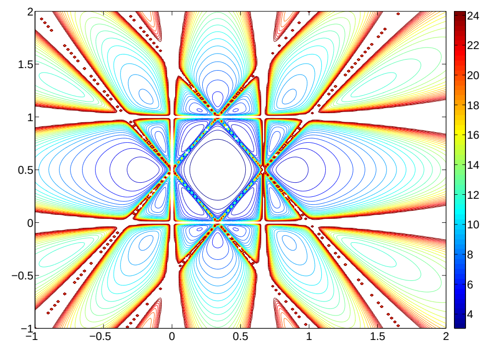
The objective function 2.15 presents a reduction of 3/4 of its original cost. Note that this is an important improvement, since all the optimization procedure is based on this function and it will be called a large number of times.

Next we repeat the examples introduced in Figures 2.6 and 2.10, using the objective function (2.15) instead of (2.10).

Figure 2.12(a) presents the optimized objective function for the quadrilateral with three fixed nodes and a fourth node free on the x-axis (Figure 2.6). Note that Figure 2.12(a) is exactly the same function that was presented in Figure 2.7(b) except from a constant.



(a)



(b)

Figure 2.12: a) Optimized objective function of $\mathbf{x}_2(x)$, $x \in [-5, 5]$, of the mesh presented on Figure 2.4. b) Optimized objective function of node 6 of the mesh presented of Figure 2.10, with a fixed threshold equal to 25.

Figure 2.12(b) shows a representation of the optimized objective function (2.15). We can compare it with Figure 2.11 in order to see that this new definition of objective function does still hold the same global minimum.

Note that the level curves are similar in both figures. The only difference is the value that the level curves take, which now differences from the original plot from just a constant.

Further on, the definition of the quadrilateral objective function can be even more optimized. The decomposition into four triangles and thus, taking as ideal triangle the isosceles one, guaranties that the ideal quadrilateral is the square.

However, note that if we decompose a quadrilateral element into three triangles (selecting the triangles that belong to the free node, as we have just done) and we impose them to be the ideal triangle, we already have the ideal quadrilateral. This indeed proves that Definition 3.18 of objective function is correct.

Moreover, if we only take into account two of the four underlying triangles considered in the decomposition of the quadrilateral element, we still can define an objective function with a behaviour similar to the objective function (2.15). That is, if we impose that the ideal triangle for the two selected triangles is the isosceles and rectangle triangle, then, the ideal quadrilateral element is the square.

The prove is straight forward: since we impose that three edges of the quadrilateral have the same length and that two angles are rectangle, the quadrilateral defined by such triangles can only be the square. This property can be really useful, since it reduces the number of required computations of the initial objective function to the half.

Thus, we could define the quadrilateral objective function as a weighting of the sum of the objective functions corresponding to two triangles, even obtaining the same results. Note that we can freely choose which triangles (from the three that have the free node as a vertex) we do include in the definition of the objective function. In order to have a more robust optimization objective function we can choose the two triangles with opposite angles, since the symmetry of such definition may help to have a more robust function for our purpose (see Figure 2.13).

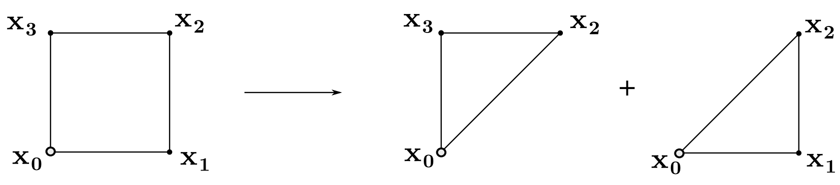


Figure 2.13: Simplification of the standard division of a square in order to compute its objective function.

This way, the objective function would be defined as:

$$\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{\eta(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) + \eta(\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_3)}{2}, \tag{2.16}$$

being $\mathbf{x}_0$ the free node.

Figures 2.14(a) and 2.14(b) present the plot of the second modification of the objective function for the presented examples. Note that this third definition gives the same global minimum but some of the asymptotes that appeared in the previous definitions have disappeared. This indicates that this objective function does not recognize some positions that define the eliminated triangle as degenerated. However, since the minimum is not modified, the fact that some asymptotes are eliminated could be indeed useful, since the resulting objective function is smoother, having less local minima.

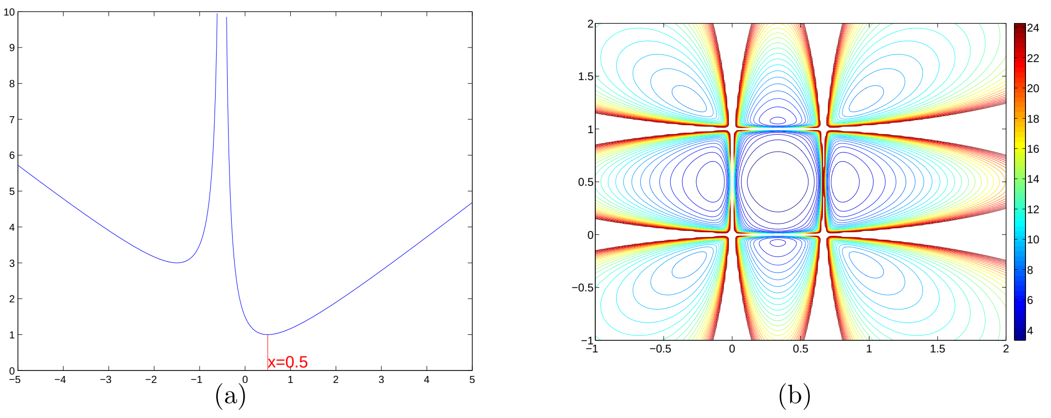


Figure 2.14: a) Second optimized objective function of $\mathbf{x}_2(x)$, $x \in [-5, 5]$, of the mesh presented on Figure 2.4. b) Second optimized objective function of node 6 of the mesh presented of Figure 2.10, with a fixed threshold equal to 25.

The objective function (2.16) avoids the computation of the objective function of a subtriangle that does vary with the optimization of the free node. However, the argument that we are using to eliminate the contribution of the third triangle is that imposing to two of the subtriangles to be isosceles rectangle is enough to have as ideal element the square.

As result we have a key that does accomplish the proposed objective in a computationally cheaper way, but that may be indeed less robust than the previous definitions. This way we cannot say anything about which of the new modifications of the original objective function will be more efficient in our optimization procedure without testing them.

From now on, we take (2.15) as the default objective function to use. However, when working with the examples, we will bring back (2.16) in order to check if the results of this second modified function are good or not compared to the ones of (2.15).

2.6 Modified objective function: a function for untangling and smoothing elements

2.6.1 Modified objective function for triangular element meshes

In this section we review a modification of the objective function developed in [Escobar 03] in order to avoid the asymptotes and the local minimum that appear using the original objective function (2.13).

Figure 2.9 shows that despite the fact that the objective function is smooth in the region that defines a valid local mesh $\mathcal{M}_v$, it has discontinuities at the boundary. This happens due to the fact that when σ_k tends to zero, η_k tends to infinity.

If we want to smooth the mesh and the free node is inside the feasible region, function (2.13) behaves well, because it is smooth in this region. But if there exist tangled elements, function (2.13) can not be used due to the existence of these asymptotes.

To avoid this drawback, according to [Escobar 03], we modify the objective function (2.13) in order to obtain a new one that is smooth all over $\mathbb{R}^2$. As stated in [Escobar 03], the new objective function will achieve its minimum near to the one of the original function.

The modification consists on replacing σ in (2.12) by

$$h(\sigma) = \frac{1}{2}\left(\sigma + \sqrt{\sigma^2 + 4\delta^2}\right), \tag{2.17}$$

where δ is an arbitrary parameter that is chosen depending on the problem (see [Escobar 03] for further details). Note that function (2.17) is a positive increasing function that verifies $h(0) = \delta$. Figure 2.15 presents a plot of $h(\sigma)$ versus σ that illustrates the properties of $h(\sigma)$.

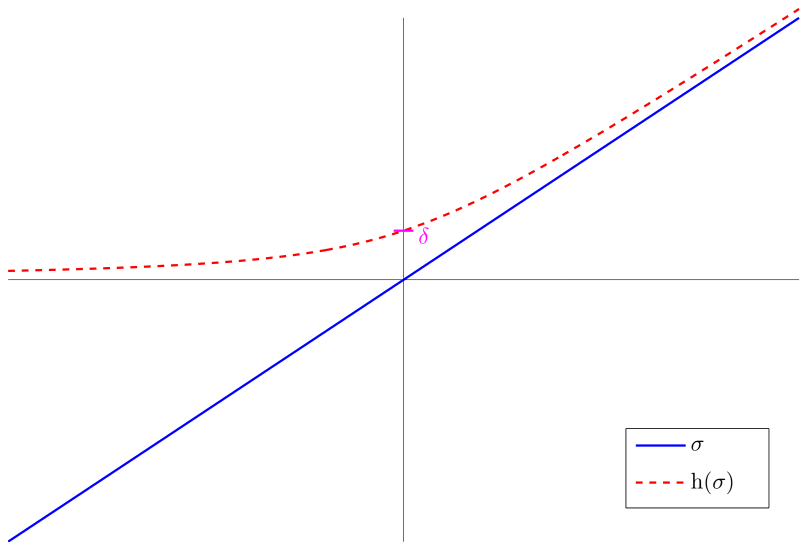


Figure 2.15: $h(\sigma)$ vs σ

Note that $\lim_{\delta\to 0} h(\sigma) = \sigma,\ \forall \sigma \geq 0$, and $\lim_{\delta\to 0} h(\sigma) = 0,\ \forall \sigma \leq 0$. Then, on the one hand, for small values of δ when the free node is in the feasible region, the modified function is similar to the original one. On the other hand, if it is in a tangled position, as it gets further from the feasible region ($\sigma \to \pm\infty$), the function never loses its smoothness but has limit equal to infinity ($\eta \overset{h(\sigma)\to 0}{\longrightarrow} \infty$). Further details on the behaviour of $h(\sigma)$ and on the selection of the value of δ can be found in [Escobar 03].

Taking into account this modification, see Equation (2.17), the new objective function for the local mesh is

$$|K_\eta^*|(\mathbf{x}) = \left(\sum_{k=1}^{m} (\eta_k^*)^p(\mathbf{x}) \right)^{1/p}, \tag{2.18}$$

where

$$\eta_k^* = \frac{|\mathbf{S}_k|}{2h(\sigma_k)}. \tag{2.19}$$

We will proceed by reproducing the already presented examples for the standard objective function, but now using the modified version. Figure 2.16 presents the modified objective function[5] of the mesh on Figure 2.4 where all the nodes of a triangle where fixed and one free node was moved on the x-axis:

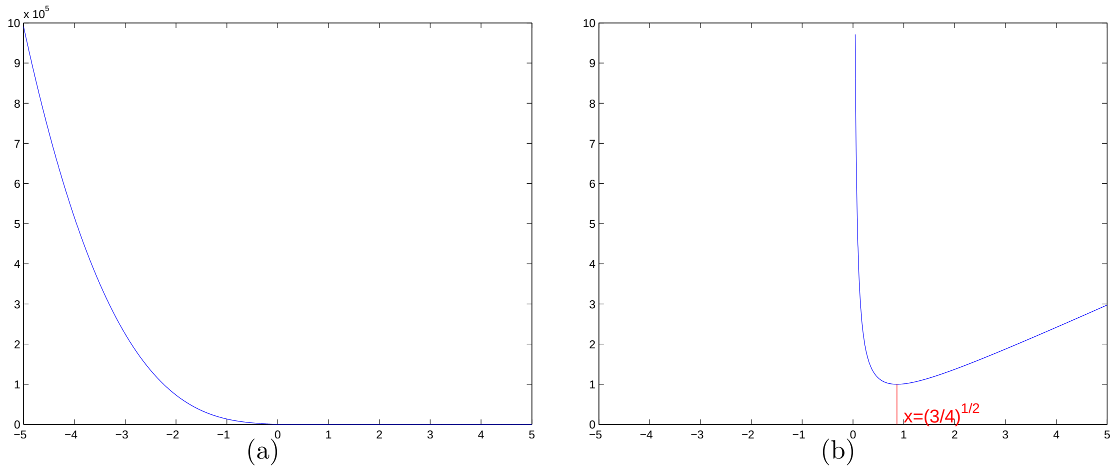


Figure 2.16: a) Modified objective function of $\mathbf{x}_2(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.4). b) Modified objective function of $\mathbf{x}_2(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.4) with a threshold equal to 10.

Note that now the asymptotes have disappeared and that the function is smooth over all the x-axis and preserves the minimum close to the original one. This would

[5]Note that Figures 2.16(a) and 2.16(b) present the same plot, but Figure 2.16(b) focuses in the behaviour of the function near the minimum.

let us find the correct configuration of the triangle if we assume that the order on the nodes of the triangle[6] is $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2\}$ (and hence, the minimum must be on the right-hand side of the domain).

Figure 2.17 presents the plots of the new objective function, Equation (2.18), for the node $\mathbf{x}_5$ of the mesh defined in Figure 2.8.

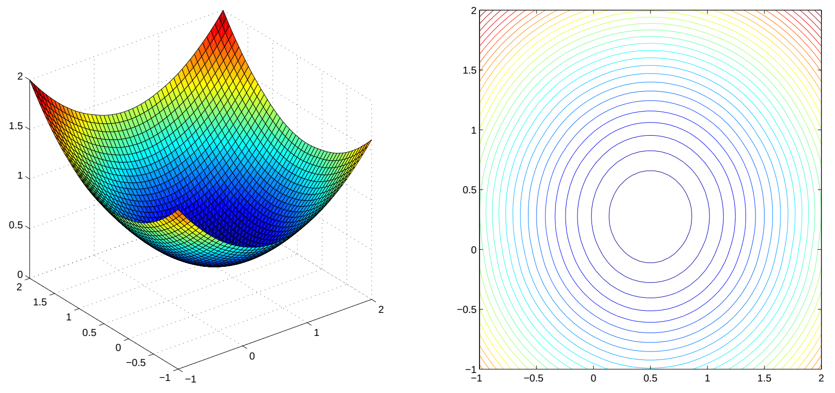


Figure 2.17: Modified objective function of node 5 of the mesh of Figure 2.8.

Note that this new function has no asymptotes and is smooth all over the domain. The minimum of the function approximately the same that the original function had in the feasible zone. Then, using this function instead of the original (2.9) will let us untangle as well as smooth the mesh.

2.6.2 Modified objective function for quadrilateral element meshes

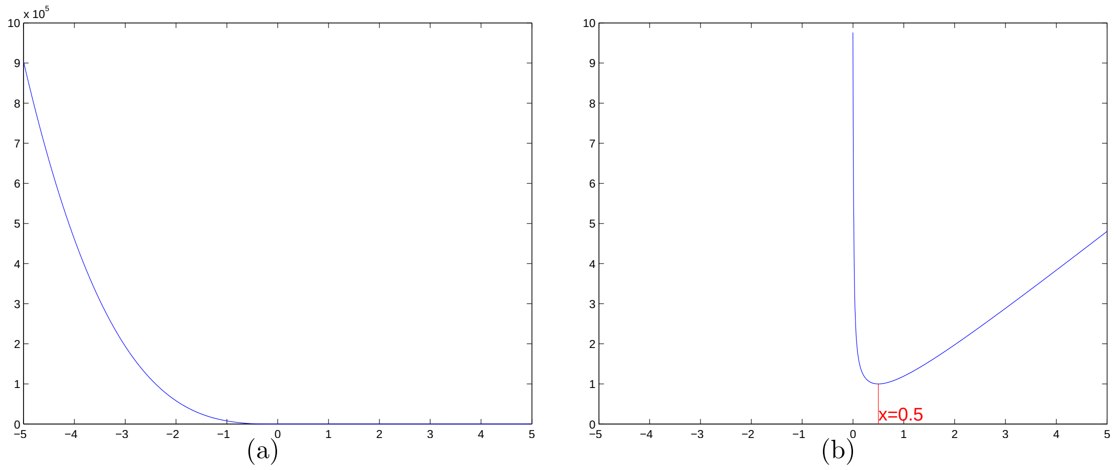
Taking into account the modification for the objective function for triangular elements introduced in Section 2.6 and taking into account the new objective function for quadrilaterals, presented in Equation (2.15), we define the modified objective function for quadrilateral elements as

$$\eta^*(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{\eta^*(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) + \eta^*(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_3) + \eta^*(\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_3)}{3}, \qquad (2.20)$$

and then, the objective function for a local mesh is defined analogously to Equation (2.18).

[6]Note that there are two possible ideal configurations, but that once we infuse an orientation and an order on the triangle -inherited from the mesh in which it is placed-, just one configuration is valid.

Figure 2.18 shows the modified objective function for the mesh presented on Figure 2.6, a quadrilateral with a free node. Note that this modified objective function has no asymptotes and through a minimization procedure we would be able to find its global minimum (the optimal configuration of the quadrilateral) starting from any initial point.



(a) (b)

Figure 2.18: a) Modified objective function of $\mathbf{x}_0(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.6). b) Modified objective function of $\mathbf{x}_0(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.6) with a threshold equal to 10.

Figure 2.19 presents the plot of the modified objective function for a submesh for the node $\mathbf{x}_6$ of the example presented in Figure 2.10:

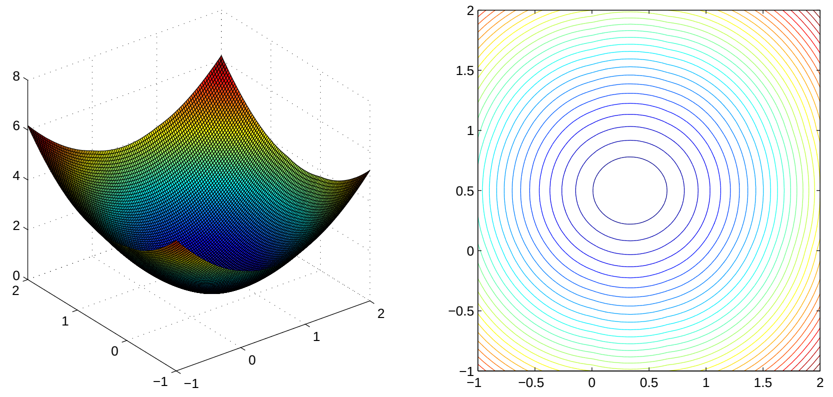


Figure 2.19: Modified objective function of node 6 of the mesh presented in Figure 2.10.

Once again, the modified objective function eliminates the asymptotes but maintains the global minimum on approximately the same location than the original objective function (2.15). Hence, the modification of the objective function allows us using a minimization method from any starting point (even if it defines a tangled configuration) ensuring that it will reach the global minimum of the objective function.

Similarly, this modification can be also applied to the objective function (2.16). In this case, the modified objective function becomes:

$$\eta^*(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{\eta^*(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) + \eta^*(\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_3)}{2}. \tag{2.21}$$

If we use the second simplification of the standard objective function, the same effect is still preserved: the remaining asymptotes that this objective function disappear and it results in a smooth function all over the domain. Moreover, note that function (2.21) presents the minimum in the same position than (2.20). Now, the asymptotes that differentiated both functions are not present, and Figures 2.18 and 2.20 become nearly equal.

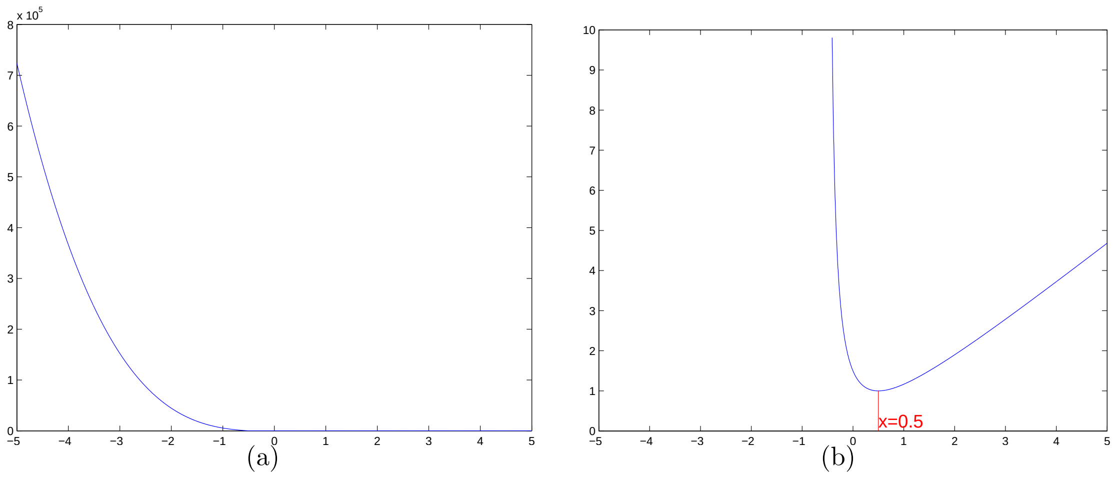


Figure 2.20: a) Modified optimized objective function of $\mathbf{x}_0(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.6). b) Modified optimized objective function of $\mathbf{x}_0(x)$, $x \in [-5, 5]$ (of the mesh presented on Figure 2.6) with a threshold equal to 10.

Figure 2.21 presents the modified optimized objective function (2.21) for the second proposed example. Note that the asymptotes have disappeared but the minimum is kept in the same position than in the standard objective function (Figure 2.11) and its modified version (Figure 2.19).

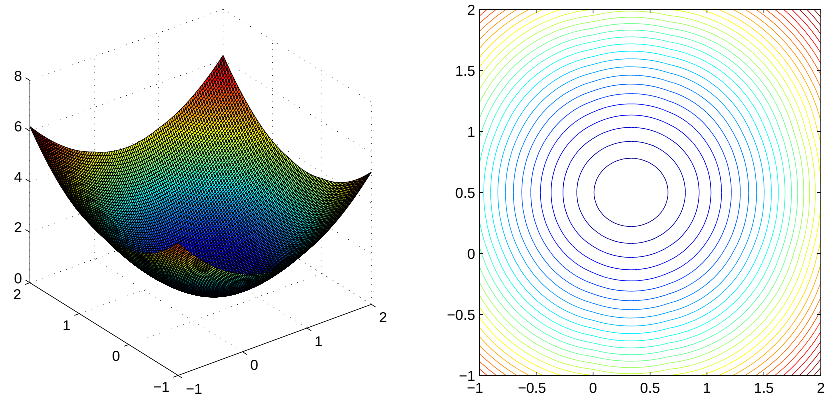


Figure 2.21: Modified objective function (2.16) of node 6 of the mesh of Figure 2.10, with a fixed threshold equal to 25.

As we commented when we presented this second optimized objective function, it seems to hold the same minimum and to have an even smoother behaviour than the original one (with three or four triangles). However, for the moment we do not consider it as the standard objective function for quadrilaterals, since we are not able to ensure that it is as robust as the standard one. In the examples, see Chapter 5, we will compare the behaviour of both presented objective functions and analyse which one is more convenient.

2.6.3 Determination of parameter δ

We have already checked the good performance of the modification of the standard objective function. Despite the good results that the use of $h(\sigma, \delta)$ presents, the modified objective function is sensitive to the selection of the δ parameter. Hence, in this section we detail how to determine the proper value of parameter δ. We detail the selection of δ for triangular meshes to simplify the exposition. The case of quadrilateral elements can be reduced to the triangular case in a straightforward manner.

To determine the right value of parameter δ we have to take into account the value σ. That is, δ has to be large enough to ensure that δ^2 is significant compared to σ^2. In addition, it has to be small enough to ensure a small perturbation of the location of the minimum of the objective function. Specifically, $h(\sigma, \delta)$ can slightly modify the location of the minimum if it ensures always that the new location is inside the feasible region. Thus, once the modified objective function has driven the free node inside the feasible region, we can use the standard objective function to obtain the exact minimum.

The modification of the objective function by means of function $h(\sigma, \delta)$ is proposed in [Escobar 03]. In this work the authors present several examples where δ has the same value for all the mesh nodes. However, we have detected that this approach does not work properly for several of our test meshes. To address this issue, we propose two different approaches to determine the value of δ.

The goal of the first approach is to obtain for each submesh the smallest δ that determines a convex objective function without asymptotes. Moreover, we require δ to be a constant in each submesh to simplify the computation of the objective function derivatives. To this end, the value of δ can only depend on the initial σ_k^0 of the $k = 1, \ldots, n$ triangles of the submesh.

We propose to use the modified objective function only if there is a tangled triangle in the analysed submesh, that is, an element with a negative σ_k^0. Moreover, we use as a reference the value σ_-^0 corresponding to the triangle with the smallest value of σ_0. In this way, we can control undesired numerical cancellations in the expression of $h(\sigma_-^0, \delta)$ that determine singularities of η^*. These numerical cancellations appear in triangle configurations where $\sigma_-^0 \ll 0$ and $0 < \delta \ll 1$ because $(\sigma_-^0)^2 + 4\delta^2 \approx (\sigma_-^0)^2$ and therefore $h(\sigma_-^0, \delta) \approx 0$.

We proceed to develop a formula to determines the proper value of δ. If we recall Figure 2.15, $h(\sigma, \delta)$ is an increasing function. In addition, in each iteration of the minimization procedure we obtain new positions of the free node that determine increasing values of σ. Thus, it is enough to ensure that $h(\sigma_-^0, \delta) > 0$ to avoid singularities arise during the minimization. Moreover, we propose to select the lowest possible value for δ that ensures $h(\sigma_-^0, \delta) \neq 0$.

Following these ideas, we can deduce the formula that determines the proper value of δ. We want

$$h(\sigma_-^0, \delta) = \frac{1}{2}\left(\sigma_-^0 + \sqrt{(\sigma_-^0)^2 + 4\delta^2}\right) = tol > 0. \tag{2.22}$$

Hence,

$$\delta(\sigma_-^0) = \frac{\sqrt{(2 \cdot tol + |\sigma_-^0|)^2 - (\sigma_-^0)^2}}{2} = \sqrt{tol^2 + tol \cdot |\sigma_-^0|}. \tag{2.23}$$

The parameter tol must be relatively smaller than σ_-^0, but large enough to not becoming zero compared to it. Thus, we should take it as

$$tol = \alpha \cdot |\sigma_-^0|, \tag{2.24}$$

where $\alpha = 10^{-n} \in (10^{-3}, 10^{-6})$. Note that now, we are controlling that

- $h(0, \delta(\sigma_-^0)) = \delta(\sigma_-^0)$, and
- $h(\sigma_-^0, \delta(\sigma_-^0)) = tol$.

Figure 2.22 presents the same plot of Figure 2.15, highlighting the points we are now able to control:

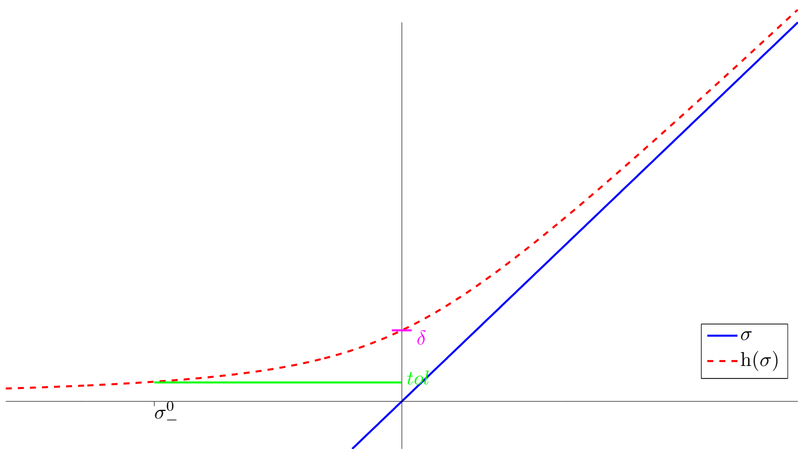


Figure 2.22: $h(\sigma, \delta(\sigma_-^0))$ vs σ

As we choose *tol* to be (2.24), expression (2.23) can be simplified:

$$\delta(\sigma_-^0) = \sqrt{tol^2 + tol \cdot |\sigma_-^0|} \overset{tol=\alpha\cdot|\sigma_-^0|}{=} |\sigma_-^0| \cdot \sqrt{\alpha^2 + \alpha}. \tag{2.25}$$

This first approach determines a formula for δ in function of the initial σ_k^0 of the analysed submesh. Following, we detail the second approach which is simpler and even more robust and effective.

The second approach is based in the fact that the presented quality metric only depends on the shape of the measured element. Specifically, the quality metric is invariant under rotations, translations and changes of scale. Hence, in this approach we propose to scale any given triangle into a certain size that fits to a constant delta.

For triangles whose side length is around one, we have empirically tested that a value $\delta = 0.01$ accomplishes the two already explained points that we want δ to hold. Hence, given a submesh we follow the next procedure:

1. Compute the lengths of the boundary edges of the triangles of the submesh.
2. Compute the maximum of such lengths, Λ.
3. Finally, scale the triangles dividing the length of the edges by Λ.

This approach is robust because δ can be determined for a reference configuration and all other configurations can be reduced to this one by scaling. In our implementation we use the value $\delta = 0.01$, that slightly perturbs the location of the minimum and ensures the convergence of the minimization process. Note that, the empirical value $\delta = 0.01$ can also be obtained from formula (2.23), since $\delta(1) \approx 0.01$. That is, 1 is a upper bound of the area of a general triangle with edges of length less or equal to 1.

Note that from now on we will use the second approach. Abusing the notation, given a submesh $\mathcal{M}$ and its triangles $t_k,\ k = 1, \ldots, m$, we define a scaling function $\tilde{\lambda}(t)$, and instead of using the standard objective function $\eta^*(t)$, use a composed version $\breve{\eta}^*(t) = \eta^* \circ \tilde{\lambda}(t)$. Since the objective function is invariant under change of scale, it holds that

$$\breve{\eta}^*(t) = \eta^* \circ \tilde{\lambda}(t) = \eta^*(t).$$

In order to simplify the notation for the following sections, we rename $\breve{\eta}^*(t)$ by the usual name η^*, since we are no more going to use the standard one (in Section 3.3 we retake this function and specify its formulation mixed with the surface theory).

It is important to point out that the second approach can only be applied when the objective function is invariant under scaling. In that case, such when the quality measure takes into account the size of the elements, we can use the first approach determined by Equation (2.23).

Chapter 3

Objective function for surfaces

3.1 Objective function for triangles on surfaces

The purpose of this chapter is to extend the smoothing and untangling algorithms for planar meshes presented in Chapter 2 to meshes defined on parametric surfaces. To this end, we will express the objective function of a triangle in function of the two variables that define the surface parametric space. Since a triangle on the surface is planar but it is immersed in $\mathbb{R}^3$, we want to achieve our goal by mapping this triangle to a geometrically equivalent triangle in $\mathbb{R}^2$ and apply the already known theory for triangles immersed in $\mathbb{R}^2$ [Knupp 01].

First, we introduce the following notation:

- The space of triangles immersed in $\mathbb{R}^3$ is:

$$\mathcal{T}_3 := \{\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3\}$$

- The space of triangles in the plane $\mathbb{R}^2$ is:

$$\mathcal{T}_2 := \{t = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2\}$$

- The space of triangles in the parametric plane $\mathcal{U}$ is:

$$\mathcal{T}_{\mathcal{U}} := \{t^{\mathcal{U}} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U}\}$$

Recall that given a triangle $t = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) \in \mathcal{T}_2$, it has been explained in Section 2.2.1 how to determine its quality and objective functions. In this section, the objective function for $t \in \mathcal{T}_2$ is renamed as η_2 and it is defined according to Equation (2.8).

3.1.1 The parametric objective function for triangles

Given a triangle $\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2) \in \mathcal{T}_3$ on a surface Σ, we can map it to a triangle with the same size and shape in $\mathbb{R}^2$. Assume, with no loss of generality, that $\mathbf{y}_0$ is the free node of the triangle.

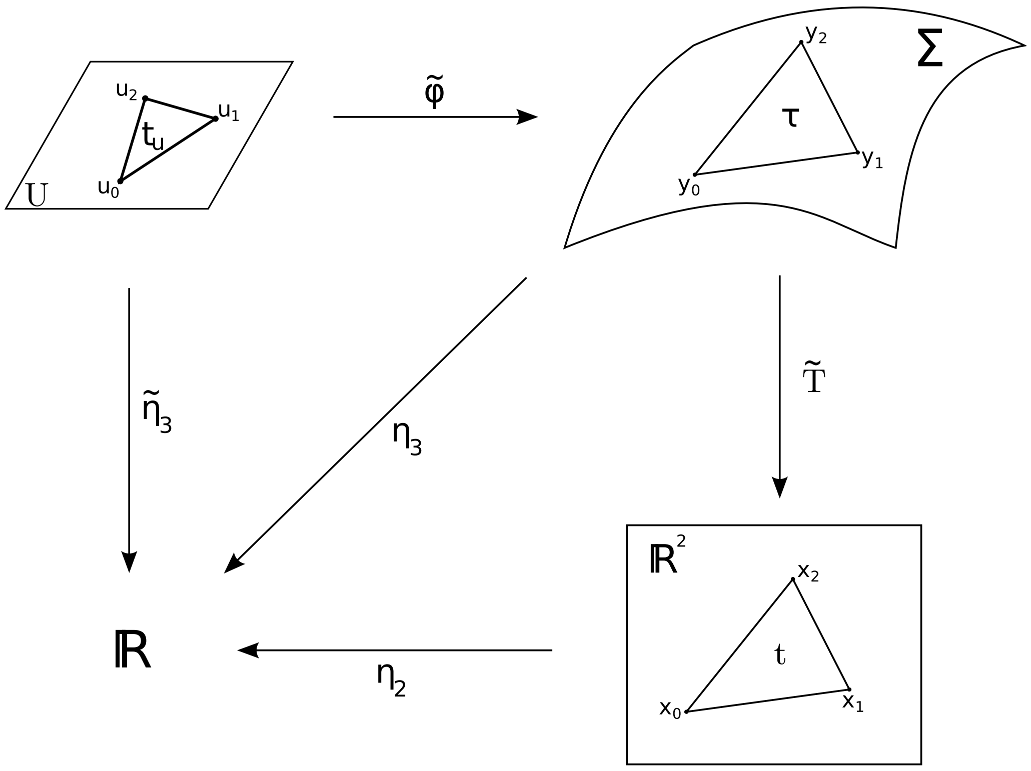


Figure 3.1: Diagram of mappings determining the parametric objective function.

Assume that the triangle $\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2)$ lies on a surface Σ that can be locally parameterized by a given function

$$\begin{array}{rccc} \varphi : & \mathcal{U} \subset \mathbb{R}^2 & \longrightarrow & \Sigma \subset \mathbb{R}^3 \\ & \mathbf{u} = (u, v) & \longmapsto & \mathbf{y} = \varphi(\mathbf{u}), \end{array} \tag{3.1}$$

that is invertible and at least $\mathcal{C}^2$. We define a new function $\tilde{\varphi}$,

$$\begin{array}{rccc} \tilde{\varphi} : & \mathcal{T}_2^{\mathcal{U}} \subset \mathcal{U} \times \mathcal{U} \times \mathcal{U} & \longrightarrow & \mathcal{T}_3 \subset \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \\ & (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2) & \longmapsto & (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2) = (\varphi(\mathbf{u}_0), \varphi(\mathbf{u}_1), \varphi(\mathbf{u}_2)), \end{array} \tag{3.2}$$

that maps a triangle $t^{\mathcal{U}} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2)$ on the parametric plane to a triangle $\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2)$ that lies on the surface defined by φ, see Figure 3.1.

Since τ is planar, we can define a mapping $\tilde{\mathbf{T}}$ (see Figure 3.1) such that

$$\begin{array}{rccc} \tilde{\mathbf{T}} : & \mathcal{T}_3 & \longrightarrow & \mathcal{T}_2 \\ & \tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2) & \longmapsto & t = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2), \end{array} \tag{3.3}$$

where τ and t are geometrically equivalent, but each one is placed on a different space. Several affine mappings can be constructed such that they map a certain triangle with three coordinate vertices to an equivalent triangle in $\mathbb{R}^2$.

Following, we define a linear mapping $\mathbf{T}$ from the canonical basis on $\mathbb{R}^3$, where the triangle τ is defined, to a new orthogonal basis obtained from a proper combination of two edges of the triangle.

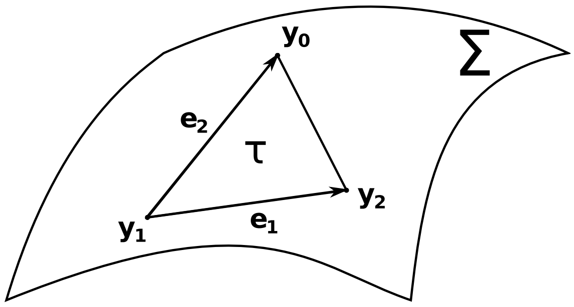


Figure 3.2: Vector edges $\mathbf{e}_1$ and $\mathbf{e}_2$ for a triangle $\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2)$ on a surface Σ.

Let

$$\begin{aligned}\mathbf{e}_1 &:= \mathbf{y}_2 - \mathbf{y}_1,\\ \mathbf{e}_2 &:= \mathbf{y}_0 - \mathbf{y}_1,\end{aligned}$$

be two vector edges of the triangle, see Figure 3.2. We define

$$\begin{aligned}\tilde{\mathbf{e}}_1 &:= \frac{\mathbf{e}_1}{\|\mathbf{e}_1\|}, \quad \text{and}\\ \tilde{\mathbf{e}}_2 &:= \text{sign} \cdot \tilde{\mathbf{e}}_{2,0}, \text{ where } \tilde{\mathbf{e}}_{2,0} = \frac{\mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1}{\|\mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1\|}\end{aligned}$$

as the two orthonormal vectors of the new basis, where *sign* is defined such that we obtain a well oriented orthonormal basis. Specifically, if we denote by $\mathbf{n}$ the normal vector to the surface at $\mathbf{y}_1$, obtained as $\mathbf{n}(\mathbf{y}_1) = \frac{\partial \varphi}{\partial u}(u_1, v_1) \times \frac{\partial \varphi}{\partial v}(u_1, v_1)$, then we define the *sign* as

$$\text{sign} := \frac{(\tilde{\mathbf{e}}_1 \times \tilde{\mathbf{e}}_{2,0}) \cdot \mathbf{n}}{|(\tilde{\mathbf{e}}_1 \times \tilde{\mathbf{e}}_{2,0}) \cdot \mathbf{n}|} = \frac{\det(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}{|\det(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})|}.$$

Now, we can define the required linear mapping

$$\begin{aligned}\mathbf{T}: \ \mathbb{R}^3 &\longrightarrow \quad \mathbb{R}^2\\ \mathbf{y} &\longmapsto \mathbf{M} \cdot (\mathbf{y} - \mathbf{y}_1)\end{aligned} \tag{3.4}$$

where $\mathbf{M}$ is the 2×3 matrix

$$\mathbf{M} = \begin{pmatrix} \tilde{\mathbf{e}}_1^T \\ \tilde{\mathbf{e}}_2^T \end{pmatrix}.$$

Note that $\mathbf{M}$ does depend on the three nodes of the considered triangle, $\mathbf{M} \equiv \mathbf{M}(\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2)$.

Now, using the linear mapping $\mathbf{T}$, we can define $\tilde{\mathbf{T}}$ for the space $\mathcal{T}_3$:

$$\begin{array}{rccc} \tilde{\mathbf{T}}: & \mathcal{T}_3 & \longrightarrow & \mathcal{T}_2 \\ & \tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2) & \longmapsto & t = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) = (\mathbf{T}(\mathbf{y}_0), \mathbf{T}(\mathbf{y}_1), \mathbf{T}(\mathbf{y}_2)). \end{array} \tag{3.5}$$

Note that

$$\mathbf{T}(\mathbf{y}_1) = \mathbf{M} \cdot (\mathbf{y}_1 - \mathbf{y}_1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \text{ and} \tag{3.6}$$

$$\mathbf{T}(\mathbf{y}_2) = \mathbf{M} \cdot (\mathbf{y}_2 - \mathbf{y}_1) = \begin{pmatrix} \frac{\mathbf{e}_1^T}{\|\mathbf{e}_1\|} \\ \text{sign} \cdot (\tilde{\mathbf{e}}_{2,0})^T \end{pmatrix} \cdot \mathbf{e}_1 = \begin{pmatrix} \|\mathbf{e}_1\| \\ 0 \end{pmatrix}. \tag{3.7}$$

Therefore, $\mathbf{T}(\mathbf{y}_1)$ and $\mathbf{T}(\mathbf{y}_2)$ are independent from node $\mathbf{y}_0$. This is of the major importance in order to simplify the final expression of the objective function for surfaces.

Finally, we compute $\mathbf{T}(\mathbf{y}_0)$:

$$\begin{aligned} \mathbf{T}(\mathbf{y}_0) &= \mathbf{M} \cdot (\mathbf{y}_0 - \mathbf{y}_1) = \begin{pmatrix} \frac{\mathbf{e}_1^T}{\|\mathbf{e}_1\|} \\ \text{sign} \cdot \left(\frac{\mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1}{\|\mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1\|} \right)^T \end{pmatrix} \cdot \mathbf{e}_2 \\ &= \begin{pmatrix} \frac{\mathbf{e}_1^T \mathbf{e}_2}{\|\mathbf{e}_1\|} \\ \text{sign} \cdot \frac{\mathbf{e}_2^T \mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1^T \mathbf{e}_2}{\|\mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1\|} \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{e}_1^T \mathbf{e}_2}{\|\mathbf{e}_1\|} \\ \text{sign} \cdot \frac{\mathbf{e}_2^T \mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1^T \mathbf{e}_2}{\sqrt{\mathbf{e}_2^T \mathbf{e}_2 - (\mathbf{e}_2^T \tilde{\mathbf{e}}_1)^2}} \end{pmatrix} \\ &= \begin{pmatrix} \|\mathbf{e}_2\| \cos(\alpha) \\ \text{sign} \cdot \sqrt{\mathbf{e}_2^T \mathbf{e}_2 - (\mathbf{e}_2^T \tilde{\mathbf{e}}_1)^2} \end{pmatrix} = \begin{pmatrix} \|\mathbf{e}_2\| \cos(\alpha) \\ \text{sign} \cdot \sqrt{\|\mathbf{e}_2\|^2 - (\|\mathbf{e}_2\| \cos(\alpha))^2} \end{pmatrix} \\ &= \begin{pmatrix} \|\mathbf{e}_2\| \cos(\alpha) \\ \text{sign} \cdot \|\mathbf{e}_2\| \sqrt{1 - \cos^2(\alpha)} \end{pmatrix} = \begin{pmatrix} \|\mathbf{e}_2\| \cos(\alpha) \\ \text{sign} \cdot \|\mathbf{e}_2\| \cdot |\sin(\alpha)| \end{pmatrix} \\ &= \begin{pmatrix} \|\mathbf{e}_2\| \cos(\alpha) \\ \|\mathbf{e}_2\| \sin(\alpha) \end{pmatrix}, \end{aligned}$$

where α is the inner angle defined between $\mathbf{e}_1$ and $\mathbf{e}_2$ (see Figure 3.3).

Therefore, the proposed mapping, is equivalent to a geometric transformation by means of trigonometry reasoning presented in Figure 3.3. Note that we define the mapping in a vectorial approach in order to avoid angle indeterminations brought up by the inverse of the trigonometry functions that are required to compute α.

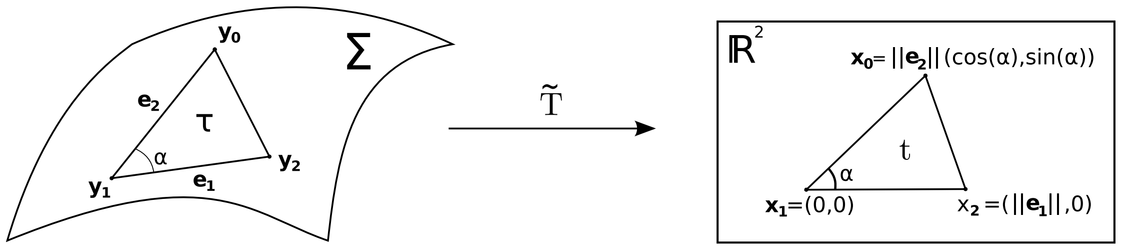


Figure 3.3: Scheme of function $\tilde{\mathbf{T}}$.

Now, we have a set of mappings that allow us to write the objective function for a triangle τ according to its parametric coordinates (see Figure 3.1):

$$\begin{array}{rcccccl} \tilde{\eta_3}: & \mathcal{T}_{\mathcal{U}} & \xrightarrow{\tilde{\varphi}} & \mathcal{T}_3 & \xrightarrow{\eta_3} & \mathbb{R} \\ & (\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2) & \longmapsto & \tilde{\varphi}(\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2) & \longmapsto & \eta_3(\tilde{\varphi}(\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2)), \end{array} \tag{3.8}$$

where

$$\begin{array}{rcccccl} \eta_3: & \mathcal{T}_3 & \xrightarrow{\tilde{\mathbf{T}}} & \mathcal{T}_2 & \xrightarrow{\eta_2} & \mathbb{R} \\ & (\mathbf{y}_0,\mathbf{y}_1,\mathbf{y}_2) & \longmapsto & \tilde{\mathbf{T}}(\mathbf{y}_0,\mathbf{y}_1,\mathbf{y}_2) & \longmapsto & \eta_2(\tilde{\mathbf{T}}(\mathbf{y}_0,\mathbf{y}_1,\mathbf{y}_2)). \end{array} \tag{3.9}$$

We denote the previous composition

$$\tilde{\eta_3} := \eta_3 \circ \tilde{\varphi} = \eta_2 \circ \tilde{\mathbf{T}} \circ \tilde{\varphi} \tag{3.10}$$

as the *parametric objective function*. We have determined a new approach for the objective function (shape index for surface triangles). As in the planar case, we can directly define the *parametric quality measure* as the inverse of the parametric objective function:

$$\tilde{q} := \frac{1}{\tilde{\eta_3}}. \tag{3.11}$$

3.1.2 The parametric objective function for an optimization procedure in a triangular mesh

The parametric objective function $\tilde{\eta_3}$ defined on Section 3.1.1 assigns a scalar value to a triangle determined by its three parametric coordinates:

$$\begin{array}{rcccccccl} \tilde{\eta_3}: & \mathcal{T}_{\mathcal{U}} \subset \mathbb{R}^6 & \xrightarrow{\tilde{\varphi}} & \mathcal{T}_3 \subset \mathbb{R}^9 & \xrightarrow{\tilde{\mathbf{T}}} & \mathcal{T}_2 \subset \mathbb{R}^6 & \xrightarrow{\eta_2} & \mathbb{R} \\ & (\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2) & \longmapsto & (\mathbf{y}_0,\mathbf{y}_1,\mathbf{y}_2) & \longmapsto & (\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2) & \longmapsto & \eta_2((\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)). \end{array}$$

Given a triangular mesh on a parametric surface, we consider the submesh $\mathcal{M}_{\mathbf{y}}$ around a node $\mathbf{y} = \mathbf{y}_0 = \varphi(\mathbf{u}_0)$. Consider a triangle $\tau = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2)$ of this submesh, in which $\mathbf{y}_0$ denotes the free node.

In this section we prove that for fixed values of $\mathbf{u}_1$ and $\mathbf{u}_2$, the function $\tilde{\eta_3}$ can be expressed as the new function

$$\begin{array}{rccccccc} \hat{\eta_3}: & \mathcal{U} \subset \mathbb{R}^2 & \xrightarrow{\tilde{\varphi}} & \Sigma \subset \mathbb{R}^3 & \xrightarrow{\tilde{\mathbf{T}}} & \mathbb{R}^2 & \xrightarrow{\hat{\eta_2}} & \mathbb{R} \\ & \mathbf{u}_0 & \longmapsto & \mathbf{y}_0 & \longmapsto & \mathbf{x}_0 & \longmapsto & \hat{\eta_2}((\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)), \end{array} \tag{3.12}$$

where $\mathbf{u}_0$ is the free node required for the optimization procedure.

To this end, we first define $\hat{\eta_3}$ as the *parametric objective function for a free node* $\mathbf{y} = \varphi(\mathbf{u})$:

$$\begin{array}{rccc} \hat{\eta_3}: & \mathcal{U} \subset \mathbb{R}^2 & \longrightarrow & \mathbb{R} \\ & \mathbf{u} & \longmapsto & \hat{\eta_3}(\mathbf{u}) := \tilde{\eta_3}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2) = \hat{\eta_2}(\tilde{\mathbf{T}}(\tilde{\varphi}(\mathbf{u}))). \end{array} \tag{3.13}$$

We will prove that

$$\hat{\eta_3}(\mathbf{u}) := \hat{\eta_2} \circ \mathbf{T} \circ \varphi(\mathbf{u}) \tag{3.14}$$

is equivalent to (3.13). Note that in (3.14) presents $\hat{\eta_3}$ as a composition of simpler functions than (3.13). Recall that $\mathbf{T}$ has been defined in Section 3.1.1 and that the following definition has been introduced:

$$\hat{\eta_2}(\mathbf{x}) := \eta_2(\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2), \tag{3.15}$$

being

$$\begin{aligned} \mathbf{x}_1 = \mathbf{T}(\mathbf{y}_1) &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ \mathbf{x}_2 = \mathbf{T}(\mathbf{y}_2) &= \begin{pmatrix} \|\mathbf{e}_1\| \\ 0 \end{pmatrix}, \end{aligned}$$

the image by $\mathbf{T}$ of the physical coordinates of the fixed vertices $\mathbf{y}_1$ and $\mathbf{y}_2$. Note that in order to be able to define the restriction (3.13) of function (3.8) through the composition of functions (3.1), (3.4) and (3.15) it has to be checked that $\mathbf{x}_1$ and $\mathbf{x}_2$ can be computed independently of the free node $\mathbf{y} = \varphi(\mathbf{u})$:

First, it is clear that $\mathbf{y}_1 = \varphi(\mathbf{u}_1)$ and $\mathbf{y}_2 = \varphi(\mathbf{u}_2)$ are independent from $\mathbf{u}_0$. Second, despite we have seen that the analytical expression of function $\mathbf{T}$, Equation (3.4), does depend on both $\mathbf{y}, \mathbf{y}_1, \mathbf{y}_2$, we have checked in (3.6) and (3.7) that the image by $\mathbf{T}$ of $\mathbf{y}_1$ and $\mathbf{y}_2$ is constant on $\mathbf{y}$.

Hence, it is correct to consider $\hat{\eta_3}(\mathbf{u})$ as objective function of the free node of a given physical 3D triangle parameterized from a parametric plane $\mathcal{U}$. Equation (3.16) proves that $\hat{\eta_3}(\mathbf{u})$ and $\tilde{\eta_3}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2)$ are indeed equivalent when $\mathbf{u}_1$ and $\mathbf{u}_2$ are fixed:

$$\begin{aligned} \hat{\eta_3}(\mathbf{u}) &= (\hat{\eta_2} \circ \mathbf{T} \circ \varphi)(\mathbf{u}) = \eta_2(\mathbf{T}(\varphi(\mathbf{u})), \mathbf{x}_1, \mathbf{x}_2) \\ &= \eta_2(\mathbf{T}(\varphi(\mathbf{u})), \mathbf{T}(\mathbf{y}_1), \mathbf{T}(\mathbf{y}_2)) = \eta_2(\tilde{\mathbf{T}}(\varphi(\mathbf{u}), \mathbf{y}_1, \mathbf{y}_2)) \\ &= \eta_2(\tilde{\mathbf{T}}(\tilde{\varphi}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2))) = (\eta_2 \circ \tilde{\mathbf{T}} \circ \tilde{\varphi})(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2) \\ &= \tilde{\eta_3}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2). \end{aligned} \tag{3.16}$$

We define the *parametric objective function* of the submesh $\mathcal{M}_{\mathbf{u}}$ of $\mathbf{u}$ as the p-norm (recall that we use $p = 2$) of the parametric objective function of all the triangles of its local submesh. Similarly to Equation (2.13), given an inner node $\mathbf{u}$ we define the parametric objective function of the submesh on node $\mathbf{u}$ as

$$K^p_{\hat{\eta_3}}(\mathbf{u}) = \left(\sum_{k=1}^{m} \left(\hat{\eta_3}_k\right)^p (\mathbf{u}) \right)^{1/p} . \tag{3.17}$$

3.2 Objective function for quadrilateral meshes on surfaces

In this section we extend the *parametric approach* for the quality and objective functions in a triangular surface mesh to a quadrilateral surface mesh. To develop the extension, we follow up the approach presented in Section 2.3 for planar elements in the plane.

First recall that the objective function of a quadrilateral is defined through a weighting of the objective functions of the triangles in which we subdivide such quadrilateral (see Figure 2.3). As we explained in Section 2.5.1, we use the first simplification of the standard objective function for quadrilaterals, see Equation (2.15). Thus, we use the three triangles adjacent to the free node to define the quadrilateral objective function. If we denote by $\tilde{\eta_3}(\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k)$ the parametric objective function of a given triangle $(\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k)$ on the parametric plane $\mathcal{U}$ (recall Figure 3.1), the parametric function for a quadrilateral surface mesh can be written as

$$\tilde{\eta_3}(\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3) := \frac{\tilde{\eta_3}(\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2) + \tilde{\eta_3}(\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_3) + \tilde{\eta_3}(\mathbf{u}_0, \mathbf{u}_2, \mathbf{u}_3)}{3}, \tag{3.18}$$

being $\mathbf{u}_0$ the free node.

Note that this is the natural extension that follows from the approach for planar quadrilaterals. However, now the objective function is determined by the parametric coordinates of the vertices of three triangles instead of their physical coordinates.

Moreover, we define the restricted parametric objective function for quadrilaterals as

$$\hat{\eta_3}(\mathbf{u}_0) := \frac{\hat{\eta_3}^{\mathbf{u}_1,\mathbf{u}_2}(\mathbf{u}_0) + \hat{\eta_3}^{\mathbf{u}_1,\mathbf{u}_3}(\mathbf{u}_0) + \hat{\eta_3}^{\mathbf{u}_2,\mathbf{u}_3}(\mathbf{u}_0)}{3}, \tag{3.19}$$

where $\hat{\eta_3}^{\mathbf{u}_i,\mathbf{u}_j}(\mathbf{u}_0)$ denotes the restricted objective function for the triangle defined by the free node $\mathbf{u}_0$ and the fixed vertices $\mathbf{u}_i$ and $\mathbf{u}_j$.

We have also to extend the parametric objective function for quadrilaterals to local meshes of a given surface node. This extension will allow to smooth the surface nodes. Thus, as in the previous section for the triangular case, given a submesh $\mathcal{M}_{\mathbf{u}}$ of an inner node $\mathbf{u}$, we define the *parametric objective function* of the submesh as:

$$K^p_{\hat{\eta_3}}(\mathbf{u}) := \left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^p (\mathbf{u}) \right)^{1/p}, \tag{3.20}$$

where $\hat{\eta_3}_k$ is the restricted parametric objective function for the kth quadrilateral element of the submesh.

3.3 Final formulation of the parametric objective function

In this section, we present the final formulation of the objective function based on the previously developed ideas. This objective function allows to perform the quality optimization process for meshes on parameterized surfaces. We just specify the objective function for a triangle, since the extension to quadrilaterals and submeshes is straightforward from the triangle expression.

Given a triangle t expressed in parametric coordinates as $(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2)$, where $\mathbf{u}$ are the coordinates of the free node, the parametric objective function of the free node is:

$$\hat{\eta_3}(\mathbf{u}) = \hat{\eta_2} \circ \lambda \circ \mathbf{T} \circ \varphi(\mathbf{u}), \tag{3.21}$$

where functions $\hat{\eta_2}$, λ, $\mathbf{T}$, and φ were presented in previous sections. Recall that function λ scales the triangles in order to use a fixed δ, see Section 2.6.3.

In order to simplify the expressions that evolve from the planar case to the surface one, we have obviated λ function until now. However, lets give now a precise formulation of this auxiliary function. As we explained in Section 2.6.3, λ function determines the lengths of the external edges of the triangles of the submesh, takes the maximum of such lengths and finally scales the triangles dividing by this maximum length. Note that the required information for λ function can be obtained from the computation of function $\mathbf{T}$ without additional computational cost. Among other calculus that $\mathbf{T}$ function requires, we obtain the norm of the vectors that define the external edges: $\|\mathbf{e}_1^k\|,\ k = 1, \ldots, n$.

We define function $\tilde{\lambda}$ for a triangle t in the plane as

$$\begin{array}{rccl} \tilde{\lambda}: & \mathcal{T}_2 & \longrightarrow & \mathcal{T}_2 \\ & t = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) & \longmapsto & t_\Lambda = \frac{1}{\Lambda}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2), \end{array} \tag{3.22}$$

where

$$\Lambda = \max_{k=1,\ldots,n} \|\mathbf{e}_1^k\| \tag{3.23}$$

is the maximum of the external length of the edges of the $k = 1, \ldots, n$ triangles that compose the submesh.

As we did for the objective function, we define the restricted scaling function for the free node as

$$\begin{array}{rccc} \lambda: & \mathbb{R}^2 & \longrightarrow & \mathbb{R}^2 \\ & \mathbf{x} & \longmapsto & \frac{1}{\Lambda} \cdot \mathbf{x}. \end{array} \tag{3.24}$$

Moreover, note that function $\hat{\eta_2}(\mathbf{x}) = \eta_2(\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2)$ defined on Equation (3.15) has to be redefined as $\hat{\eta_2}(\mathbf{x}) = \eta_2(\mathbf{x}, \frac{\mathbf{x}_1}{\Lambda}, \frac{\mathbf{x}_2}{\Lambda})$, in order to include the scaling of the constant vertices.

Following a similar reasoning to the one presented in Section 3.1.2, we check that the restriction (3.21) is equivalent to the expression for a whole triangle with two fixed vertices:

$$\begin{array}{rcl} \hat{\eta_3}(\mathbf{u}) & := & \hat{\eta_2} \circ \lambda \circ \mathbf{T} \circ \varphi(\mathbf{u}) = \eta_2(\lambda(\mathbf{T}(\varphi(\mathbf{u})), \frac{\mathbf{x}_1}{\Lambda}, \frac{\mathbf{x}_2}{\Lambda})) \\ & = & \eta_2(\tilde{\lambda}(\mathbf{T}(\varphi(\mathbf{u})), \mathbf{T}(\mathbf{y}_1), \mathbf{T}(\mathbf{y}_2))) = \eta_2(\tilde{\lambda}(\tilde{\mathbf{T}}(\varphi(\mathbf{u}), \mathbf{y}_1, \mathbf{y}_2)) \\ & = & \eta_2(\tilde{\lambda}(\tilde{\mathbf{T}}(\tilde{\varphi}(\mathbf{u}), \mathbf{u}_1, \mathbf{u}_2)) = \tilde{\eta_3}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2). \end{array} \tag{3.25}$$

In order to perform the optimization procedure with the final formulation of the *restricted parametric objective function* $\hat{\eta_3}$, we need to compute its derivatives with respect to the parametric variables $\mathbf{u} = (u, v)$:

$$D\hat{\eta_3}(\mathbf{u}) = D(\hat{\eta_2} \circ \lambda \circ \mathbf{T} \circ \varphi)(\mathbf{u}) = D\hat{\eta_2}|_{\lambda(\mathbf{T}(\varphi(\mathbf{u})))} \cdot D\lambda|_{\mathbf{T}(\varphi(\mathbf{u}))} \cdot D\mathbf{T}|_{\varphi(\mathbf{u})} \cdot D\varphi|_{\mathbf{u}}.$$

All the analytical expressions of $D\hat{\eta_2}$, $D\lambda$, $D\mathbf{T}$ and $D\varphi$ can be found on the Appendices A.1. Since $\hat{\eta_3}$ is a function from $\mathbb{R}^2$ to $\mathbb{R}$, its gradient is a vector 1×2. We also detail the sizes of the rest of Jacobian matrices

$$\underbrace{D\hat{\eta_3}|_{\mathbf{u}}}_{1\times 2} = \underbrace{D\hat{\eta_2}|_{\mathbf{T}(\varphi(\mathbf{u}))}}_{1\times 2} \cdot \underbrace{D\lambda|_{\mathbf{T}(\varphi(\mathbf{u}))}}_{2\times 2} \cdot \underbrace{D\mathbf{T}|_{\varphi(\mathbf{u})}}_{2\times 3} \cdot \underbrace{D\varphi|_{\mathbf{u}}}_{3\times 2},$$

since they are helpful for implementation purposes.

Chapter 4

Mesh quality optimization

In this chapter we present the main skeleton of the programs for the optimization of the parametric objective function for both quadrilateral and triangular meshes. We will start describing the structure of the smoother that we have developed using the presented theory. Afterwards, we will analyse the minimization algorithms that we have implemented in order to perform the numerical minimization of the objective function.

4.1 Mesh quality optimization algorithm

In the previous sections we have developed the parametric objective function. Specifically, given a node and its local submesh in a surface, we are able to quantify the quality of the submesh depending on the position of the inner node. Now, we want to use it in order to smooth meshes on parameterized surfaces.

Given an initial mesh, we optimize the position of each inner node by minimizing the objective function of the corresponding submesh. Once we obtain the optimal position, the mesh is updated before the position of the next nodes is optimized. The program iterates through all the nodes of the mesh several times. The process stops when the maximum displacement divided by the minimum external edge is smaller than a prescribed tolerance for all the mesh nodes. Algorithm 4.1 presents this scheme.

Algorithm 4.1 Optimization Algorithm

```
Ensure: Mesh $\mathcal{M}$.
 1: function SmoothMesh(Mesh $\mathcal{M}$)
 2:     $E \leftarrow \infty$;
 3:     $nnodes \leftarrow$ number of inner nodes of $\mathcal{M}$;
 4:     while $E > tol$ do
 5:         $E \leftarrow 0$;
 6:         for $k = 1 : nnodes$ do
 7:             $\mathbf{u}_k \leftarrow$ initial parametric coordinates of the $kth$ node;
 8:             $\mathbf{y}_k^0 \leftarrow \varphi(\mathbf{u}_k)$, physical coordinates of the $kth$ node;
 9:             $(\mathbf{u}_k, \mathbf{y}_k) \leftarrow$ SetInitialPoint(Submesh $\mathcal{M}_{\mathbf{u}_k}$);
10:             $\mathbf{u}_k \leftarrow$ OptimizeSubmesh($\mathbf{u}_k$, Submesh $\mathcal{M}_{y_k}$);
11:             $\mathbf{y}_k = \varphi(\mathbf{u}_k)$;
12:             $E_r = \frac{\|y_k^0 - y_k\|}{\text{min edge submesh } \mathcal{M}_{y_k}}$
13:             $E \leftarrow \max(E, E_r)$;
14:         end for
15:     end while
16: end function
```

During the execution of the inner loop on the nodes, two functions are called: SetInitialPoint and OptimizeSubmesh.

Note that the optimal position of the inner node does not depend on its initial location. However, if the initial location of the inner point is close to the optimum, the minimization process requires less iterations. In this way, procedure SetInitialPoint is responsible to choose a good guess of the optimal location to use as initial point for the minimization process.

Specifically, procedure SetInitialPoint selects the initial point from among two candidates. The first candidate is the current location of the node. The second candidate is the geometrical centre of the external nodes of the submesh. Then, procedure SetInitialPoint computes the value of the objective function for both candidates. Finally, the candidate point with the lowest value of the objective function is selected as the initial point of the minimization process. It is important to point out that the geometrical centre is always inside the convex hull[1] of the external nodes. Thus, if these nodes determine a convex set, the geometrical centre is inside the feasible region, it determines an untangled configuration and it is close to the minimum of the objective function. Function SetInitialPoint is detailed in Algorithm 4.2.

[1]The convex hull or convex envelope for a set of points X in a real vector space V is the minimal convex set containing X.

Algorithm 4.2 Initial point for a submesh

```
Ensure: $(\mathbf{u}^i, \mathbf{y}^i)$.
 1: function setInitialPoint(Submesh $\mathcal{M}_{\mathbf{u}^0}$)
 2:     $f^0 \leftarrow \tilde{\eta}(\mathbf{u}^0; \mathcal{M}_{\mathbf{u}^0})$;
 3:     $\mathbf{u}^c \leftarrow$ assign the geometrical centre of the nodes adjacent to $\mathbf{u}^0$;
 4:     $f^c \leftarrow \tilde{\eta}(\mathbf{u}^c; \mathcal{M}_{\mathbf{u}^c})$;
 5:     if $f^c > f^0$ then
 6:         $\mathbf{u}^i = \mathbf{u}^0$;
 7:     else
 8:         $\mathbf{u}^i = \mathbf{u}^c$;
 9:     end if
10:     $\mathbf{y}^i = \varphi(\mathbf{u}^i)$;
11: end function
```

The second subfunction called in Algorithm 4.1, OptimizeSubmesh, gets a submesh and the parametric coordinates of the inner node, constructs the objective function, and brings up the minimization procedure.

We want to highlight that the performance of the implementation of the Algorithm 4.1 depends on the selected programming language. We have selected MATLAB to develop and implement the prototype of the algorithm. Thus, to enhance the performance we use a vectorized implementation instead of a loop based implementation. To this end, we have substituted the loops by vectorized expressions and we have reorganized the data. Note that non-vectorized implementations perform a loop for each inner node. In each step of the loop, the coordinates of the surrounding nodes (determined by the connectivity matrix T) and the current nodes are used to compute and optimize the value of the objective function. Recall that the coordinates matrix is a $n \times nnodes$ array, being n the dimension of the space ($n = 3$ for us), and *nnodes* the number of nodes of the matrix. These standard implementations perform very poorly in MATLAB because loops are interpreted and not compiled.

To overcome the low performance of non-vectorized implementations in MATLAB, we propose to write code that deals with several nodes at the same time. To achieve this goal, we first have to group the nodes in independent sets. That is, two nodes that are adjacent must not be moved at the same time. Specifically, we colour the mesh ensuring that all pairs of adjacent nodes have different colours (note that any surface graph can be coloured with four or less colours). Hence, all the nodes that have the same colour are not adjacent and can be optimized at the same time. In addition, we pre-compute a multi-array with all the adjacent nodes for a given node. This multi-array can be re-used through the minimization process to query the coordinates of the nodes that compose the submeshes. As this multi-array is pre-computed and re-used, the performance of the final implementation is improved. In the vectorized implementation we use a four-way array with the following ordering of the indices:

- the first index denotes the inner node of a submesh,
- the second index denotes the adjacent elements to the inner node of a submesh,
- the third index denotes the physical coordinates of a node (three coordinates in our case),
- and the fourth index denotes the nodes for a given element of a submesh (three nodes for triangles, four for quadrilaterals).

However, if we just subdivide the mesh into colours, some multi-array entries are wasted storing empty values. That is, not all the nodes of the same colour have a submesh with the same number of elements. Therefore, if we store all the coordinates in a static multi-array, the dimension for the second index has to be the maximum number of adjacent elements for all the nodes of a given colour. This results in a wasted memory storage corresponding to the submeshes that have less elements. To overcome this issue, we propose to decompose the mesh in groups of nodes with the same number of adjacent elements. Then, each one of this groups is coloured. Lets summarize the advantages of this approach:

- The classification of the nodes in adjacency groups permits to have data structured in multi-arrays that are dense (there are no zeros on the array) and static (the size of the different components of the multi-array is known). Hence, this results in a faster implementation.
- The sub-classification in colours permits that several nodes are optimized at the same time. Hence, we substitute slow interpreted loops by vectorized code. Moreover, the colouring of the mesh is the first step to the parallel implementation of the code we want to carry in the near future.

Taking into account above advantages, the initial Algorithm 4.1 is turned into Algorithm 4.3. For the implementation, it is not enough to change the structure of the program. We have also to convert standard sequential expressions to vectorized ones.

Algorithm 4.3 Restructured Optimization Algorithm

```
Ensure: Mesh $\mathcal{M}$.
 1: function SmoothMesh(Mesh $\mathcal{M}$)
 2:     $d \leftarrow \infty$;
 3:     $classAdjColour$, $numberAdjacencies$ ←GroupByClasses(Mesh $\mathcal{M}$);
 4:     while Stopping criteria not achieved do
 5:         for $a = 1, \dots, numberAdjacencies$ do
 6:             $numberColours$, $nodesClass$ ← ExtractInfo($classAdjColour, a$);
 7:             for $c = 1, \dots, numberColours$ do
 8:                 $nodesColour$ ← ExtractNodesColour(nodesClass, c);
 9:                 $\mathbf{U}$ ←parametric coordinates of the inner nodes;
10:                 $\mathbf{X}$ ←ExtractMultimatrix($nodesColour$);
11:                 $\mathbf{U}$ ← OptimizeSubmesh(Matrix $U$, 4D Matrix $X$);
12:             end for
13:         end for
14:     end while
15: end function
```

Finally, we can use two different stopping criteria, for a given tolerance *tol*:

1. Either, we can check over the relative displacements of the nodes:

$$E_r(\mathbf{Y}) = \frac{\|\mathbf{Y} - \mathbf{Y_0}\|}{\|\mathbf{e}_{min}\|} < tol_u,$$

   where $\mathbf{Y} = \varphi(\mathbf{U})$ and $\mathbf{e}_{min}$ is the length of the shortest edge in the submesh.

2. Or, we can check over the relative error of the value of the objective function:

$$E_r(\mathbf{K}) = \frac{|\tilde{\eta}(\mathbf{U}) - \tilde{\eta}(\mathbf{U_0})|}{|\tilde{\eta}(\mathbf{U})|} < tol_\eta.$$

Both criteria are valid and will terminate correctly the procedure. However, in our tests the first criteria, checking on the displacements, has proved more effective. Hence, in all the presented examples of this work we have used the stopping criteria based on the displacements and not on the value of the objective function.

4.2 Minimization methods

Our quality optimization procedure is based on the minimization of the objective function. The minimum of such function gives the position of the central node of the submesh that results in a better configuration of the elements that compose it.

However, it has not been possible for us to find a general analytical expression of the global minimum.

Hence, numerical minimization methods have to be used to find the minimum of the parametric objective function. We have used several numerical methods to find which one fits better the properties of the objective function.

The numerical methods are an auxiliary (but essential) key for our objective. Despite our purpose is not to extend too much on this section, we will take a quick look to the methods that we have chosen in this thesis: the Newton's Method and a set of line search methods (Steepest Descent Method, BFGS and Line Search Newton with Modification).

Note that we will not specify for each method which stopping criteria that has been applied, since we have established two common conditions for all of them. Until both

- condition 1: $\frac{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|}{\|\mathbf{x}_k\|} < tol_x$, and
- condition 2: $\eta(\mathbf{x}_k) = f(\mathbf{x}_k) < tol_f$,

do not hold, the minimization will continue (until a fixed maximum number of iterations has been achieved).

All the methods require the computation of the first or second derivatives. The computation of the derivatives of the objective function can be found on Appendix A.1.

4.2.1 Newton's Method

The Newton's method is a numerical minimization method that computes the new position advancing towards the so called Newton direction:

$$\mathbf{p}_\mathbf{k}^\mathbf{N} = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k), \tag{4.1}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_\mathbf{k}^\mathbf{N}. \tag{4.2}$$

The Newton direction is derived from the second-order Taylor series approximation to $f(\mathbf{x}_k + \mathbf{p})$,

$$f(\mathbf{x}_k + \mathbf{p}) \approx f_k + \mathbf{p}^T \nabla f_k + \frac{1}{2}\mathbf{p}^T \nabla^2 f_k \mathbf{p} \overset{def}{=} m_k(\mathbf{p}).$$

Assuming that $\nabla^2 f_k$ is positive definite, the Newton direction is obtained by finding the vector $\mathbf{p}$ that minimizes $m_k(\mathbf{p})$. Simply setting the derivative of $m_k(\mathbf{p})$ equal to zero, the explicit formula (4.1) is obtained.

We present the scheme for Newton's method in Algorithm 4.4:

Algorithm 4.4 Newton's Method

```
Ensure: Vector $\mathbf{x}_k$.
 1: function NewtonMethod(Vector $\mathbf{x}_0$, Mesh $\mathcal{M}$)
 2:     Fix a tolerance $tol$;
 3:     Set $\delta\mathbf{x}_0 = -\nabla^2\mathbf{f_0}^{-1}\nabla\mathbf{f_0}$;
 4:     $\mathbf{x_1} \leftarrow \mathbf{x}_0 + \delta\mathbf{x}_0$;
 5:     $k \leftarrow 1$;
 6:     while Convergenve not achieved do
 7:         Set $\delta\mathbf{x}_k = -\nabla^2\mathbf{f_k}^{-1}\nabla\mathbf{f_k}$;
 8:         $\mathbf{x_{k+1}} \leftarrow \mathbf{x}_k + \delta\mathbf{x}_k$;
 9:         $k \leftarrow k + 1$;
10:     end while
11: end function
```

Newton's method is expensive from the computational point of view, since in each iteration we have to compute the Hessian matrix and to solve a linear system. Given a good initial point, the Newton's direction $\delta x = \mathbf{p^N}$ is a descent direction, and N-R method has quadratic convergence (see [Nocedal]). However, to ensure that δx is a descent direction, the Hessian matrix $\nabla^2 f$ has to be positive definite. In general, this is only achieved near the minimum. But in our case, since the objective function is convex, $\nabla^2 f$ is always positive, and so N-R always has quadratic convergence.

4.2.2 Line search methods

The *line search strategies* choose a direction $\mathbf{p_k}$ and search along this direction from the current point $\mathbf{x}_k$ in order to find a new position in which the function takes a lower value. We move in this direction a distance α_k that has to be determined. The new position is computed as

$$\mathbf{x_{k+1}} = \mathbf{x}_k + \alpha_k \mathbf{p_k}. \tag{4.3}$$

Note that several different advancing directions $\mathbf{p_k}$ can be chosen. Each direction generates a different method, each one with different computational costs and convergence velocities. In Chapter 5 we analyse the behaviour of each one of the methods that we use in this work. Except from the modified Newton's method, none of following methods do require second derivatives, and reduce the computational cost. We now present the different methods that have been tested for the minimization procedure:

- *Steepest descent method* (see [Nocedal]): this is the simplest line search method, using the line search strategy in the steepest descent direction:

$$\mathbf{p_k} = -\nabla\mathbf{f_k^T}. \tag{4.4}$$

This is the simplest method used in this work. It has the disadvantage that depending on the behaviour of the objective function, the convergence of the method can slow. Further discussion can be found in [Nocedal].

Algorithm 4.5 presents the programming scheme for the Steepest Descent Method. We realize that it is indeed straightforward to implement and thus, it was the first method we used to carry out the minimization.

Algorithm 4.5 Steepest Descent Method

```
Ensure: Vector $\mathbf{x}_k$.
 1: function SteepestDescentMethod(Vector $\mathbf{x}_0$, Mesh $\mathcal{M}$)
 2:     Fix a tolerance $tol$;
 3:     Set $\mathbf{p_0} = -\nabla\mathbf{f_0^T} \equiv -\left(\nabla\mathbf{f}(\mathbf{x}_0)\right)^T$;
 4:     $\alpha_0 \leftarrow$FindStepLength(Vector $\mathbf{x}_0$, Vector $\mathbf{p_0}$, Mesh $\mathcal{M}$);
 5:     $\mathbf{x_1} \leftarrow \mathbf{x}_0 + \alpha_0\mathbf{p_0}$;
 6:     $k \leftarrow 1$;
 7:     while Convergenve not achieved do
 8:         Set $\mathbf{p_k} = -\nabla\mathbf{f_k^T}$;
 9:         $\alpha_k \leftarrow$FindStepLength(Vector $\mathbf{x}_k$, Vector $\mathbf{p_k}$);
10:         $\mathbf{x_{k+1}} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{p_k}$;
11:         $k \leftarrow k+1$;
12:     end while
13: end function
```

- BFGS: the *Broyden-Fletcher-Goldfarb-Shanno* method is the most popular quasi-Newton method.

  The main idea of the BFGS is to approximate of the inverse of the Hessian in order to avoid the high cost of the computation of the Hessian and afterwards solve the required linear system. It uses the following update formula to compute the inverse of the Hessian:

$$\mathbf{H_{k+1}} = (\mathbf{I} - \rho_k\mathbf{s_k}\mathbf{y_k}^T)\mathbf{H_k}(\mathbf{I} - \rho_k\mathbf{y_k}\mathbf{s_k}^T) + \rho_k\mathbf{s_k}\mathbf{s_k}^T, \tag{4.5}$$

  where

$$\begin{aligned} \mathbf{s_k} &= \mathbf{x}_{k+1} - \mathbf{x}_k, & (4.6)\\ \mathbf{y_k} &= \nabla f_{k+1} - \nabla f_k, & (4.7)\\ \rho_k &= \frac{1}{\mathbf{y_k}^T\mathbf{y_k}}. & (4.8) \end{aligned}$$

  The initial matrix $\mathbf{H_0}$ is often set to some multiple of the identity, but there is no good general strategy for choosing the multiple. We use a quite effective heuristic that consists on scaling the starting matrix after the first step has been computed but before the first BFGS update is performed (see [Nocedal]). Then, we change the default $\mathbf{H_0} = \mathbf{I}$ by setting

$$\mathbf{H_0} = \frac{\mathbf{y_k}^T \cdot \mathbf{s_k}}{\mathbf{y_k}^T \cdot \mathbf{y_k}}\mathbf{I}.$$

Algorithm 4.6 presents the programming scheme for the BFGS Method.

Algorithm 4.6 BFGS Method

```
Ensure: Vector $\mathbf{x}_k$.
 1: function BFGS(Vector $\mathbf{x}_0$, Mesh $\mathcal{M}$)
 2:     Fix a tolerance $tol$;
 3:     Compute $\mathbf{H_0}$;
 4:     $k \leftarrow 0$;
 5:     while Convergenve not achieved do
 6:         Set $\mathbf{p_k} = -\mathbf{H_k}\nabla\mathbf{f_k^T}$;
 7:         $\alpha_k \leftarrow$FindStepLength(Vector $\mathbf{x}_k$, Vector $\mathbf{p_k}$);
 8:         $\mathbf{x_{k+1}} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{p_k}$;
 9:         Define $\mathbf{s_k}$, $\mathbf{y_k}$ and $\rho_k$ by means of (4.6), (4.7) and (4.8);
10:         Compute $\mathbf{H_{k+1}}$;
11:         $k \leftarrow k + 1$;
12:     end while
13: end function
```

- *Newton's Method with Hessian Modification*: as we have previously commented, away from the solution in a standard function, the Hessian matrix may not be positive definite, and so the Newton direction (4.1) may not be a descent direction. The modification that follows overcomes this difficulty.

  Despite that when we presented the Newton's Method we commented that when working with the objective function the Hessian should be positive definite in all the domain, it is worth it to implement this modification of the standard Newton's Method in order to check how this modification behaves when dealing with our minimization problem (since this modification ensures not only that the Hessian is positive definite but also that it is *positive enough* -not close to zero-).

  The presented approach obtains the step $\mathbf{p_k}$ from a linear system identical to (4.1), except that the inverse of the Hessian matrix is replaced with a positive definite approximation. The modified Hessian is obtained by adding either a positive diagonal matrix or a full matrix to the true Hessian $\nabla^2 f(\mathbf{x}_k)$. Algorithm 4.7 presents the programming scheme of the method:

Algorithm 4.7 Line Search Newton with Hessian Modification

```
Ensure: Vector $\mathbf{x}_k$.
1: function NewtonModified(Vector $\mathbf{x}_0$, Mesh $\mathcal{M}$)
2:     Fix a tolerance $tol$;
3:     $k \leftarrow 0$;
4:     while Convergenve not achieved do
5:         Compute $\mathbf{B_k} = \nabla^2 f(\mathbf{x}_2) + \mathbf{E_k}$;
6:         $\mathbf{p_k} \leftarrow -\mathbf{B_k}^{-1}\nabla f(\mathbf{x}_k)$;
7:         $\alpha_k \leftarrow$FindStepLength(Vector $\mathbf{x}_k$, Vector $\mathbf{p_k}$);
8:         $\mathbf{x_{k+1}} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p_k}$;
9:         if $\frac{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|}{\|\mathbf{x}_k\|} < tol$ then
10:            $ConvergenceAchieved = true$;
11:        end if
12:        $k \leftarrow k + 1$;
13:    end while
14: end function
```

Note that matrix $\mathbf{E_k}$ is chosen as zero when $\nabla^2 f(\mathbf{x}_k)$ is sufficiently positive definite. Otherwise, it is chosen to ensure that $\mathbf{B_k}$ is sufficiently positive definite. See [Nocedal] for more information about the Hessian modification.

Once the advancing direction is fixed using any of the previous presented approaches, it has to be decided the distance α that has to be covered from the initial position along the decided direction. According to [Nocedal], the *step length* α has to satisfy several conditions depending on the specific criteria that one wants to follow. We will use several strategies that we summarize in the following algorithms:

- Standard Line Search (for Wolfe conditions): this scheme sets the initial step length to 1 and reduces it a given scalar ρ until it holds the Wolfe conditions (see [Nocedal]):

$$f(\mathbf{x}_k + \alpha \mathbf{p_k}) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f^T(\mathbf{x}_k)\mathbf{p_k}, \tag{4.9}$$
$$\nabla f^T(\mathbf{x}_k)\mathbf{p_k} \geq c_2 \nabla f^T(\mathbf{x}_k)\mathbf{p_k}, \tag{4.10}$$

with $0 < c_1 < c_2 < 0$ (in this project $c_1 = 10^{-4}$ and $c_2 = 0.9$ as recommended on [Nocedal]). We summarize this procedure in Algorithm 4.8.

Algorithm 4.8 Line Search for complete Wolfe conditions

```
Ensure: Real $\alpha_k$.
1: function LineSearch(Vector $\mathbf{x}_k$, Vector $p_k$)
2:     Fix $\alpha > 0,\ \rho \in (0,1),\ 0 < c_1 < c_2 < 1$;
3:     while (4.9) and (4.10) don't hold do
4:         $\alpha \leftarrow \rho\alpha$;
5:     end while
6:     $\alpha_k \leftarrow \alpha$;
7: end function
```

Note that ρ is the factor we use to decrease the step length if it is too big and the computed point does not hold the Wolfe conditions. This scalar can take different values, but for our project we have fixed it to $\rho = \frac{1}{2}$.

- Backtracking Line Search: this approach is indeed a simplification of Wolve conditions (see [Nocedal] for details). We summarize this procedure in Algorithm 4.9.

Algorithm 4.9 Backtracking Line Search

```
Ensure: Real $\alpha_k$.
1: function BackLineSearch(Vector $\mathbf{x}_k$, Vector $p_k$)
2:     Fixed $\alpha > 0,\ \rho \in (0,1),\ c \in (0,1)$;
3:     while $f(\mathbf{x}_k + \alpha\mathbf{p_k}) > f(\mathbf{x}_k) + c\alpha\nabla\mathbf{f_k^T}\mathbf{p_k}$ do
4:         $\alpha \leftarrow \rho\alpha$;
5:     end while
6:     $\alpha_k \leftarrow \alpha$;
7: end function
```

The variables that determine the backtracking line search have to be fixed. We inherit from the Wolfe conditions that the variable c has to be quite small, and thus, we have taken $c = 10^{-4}$ as it is proposed on [Nocedal]. In addition we fix $\rho = \frac{1}{2}$ as we did in Algorithm 4.8. Note that both Algorithms 4.8 and 4.9 have the exact same structure, and that the differences just lie on the stopping conditions.

Chapter 5

Examples

In this chapter several examples are presented in order to assess the properties of the proposed method. In this work, all algorithms have been implemented using MATLAB. In all the examples, the parametric surfaces are represented by NURBS.

We present four parametric surfaces with both triangular and quadrilateral meshes. For the triangular meshes we have presented three figures for each example. First, we plot an initial triangular mesh. In the second figure we present a mesh with the same topology than the initial one, but with a high number of tangled elements. In order to highlight the capabilities of the developed method, the tangled mesh is the input of the smoothing-untangling algorithm. Then, in the third figure we present the result of smoothing the tangled mesh. For the quadrilateral case we present two figures with smoothed meshes. The first figure presents the mesh that results from smoothing using the objective function (2.15) and the second one using the simplified objective function (2.16). Note that in all the figures the elements are coloured according to their quality, from blue (quality 0) to red (quality 1).

For all the examples we provide the number of nodes and elements, and statistical information about the quality of the elements of the mesh. In particular, we compute for the initial, the tangled and the optimized mesh, the following quality statistics:

- Minimum quality value over all the elements of the mesh.
- Maximum quality value over all the elements of the mesh.
- Mean value of the quality of the elements of the mesh.
- Standard deviation of the quality of the elements of the mesh.
- Number of tangled elements of the mesh.

Finally, we analyse several computational aspects of the proposed smoothing method. In particular, for each example we provide the following information:

- Number of iterations of the global algorithm.
- Elapsed time for the minimization process.
- Ratio elapsed time per iteration.
- Ratio elapsed time per iteration and node.

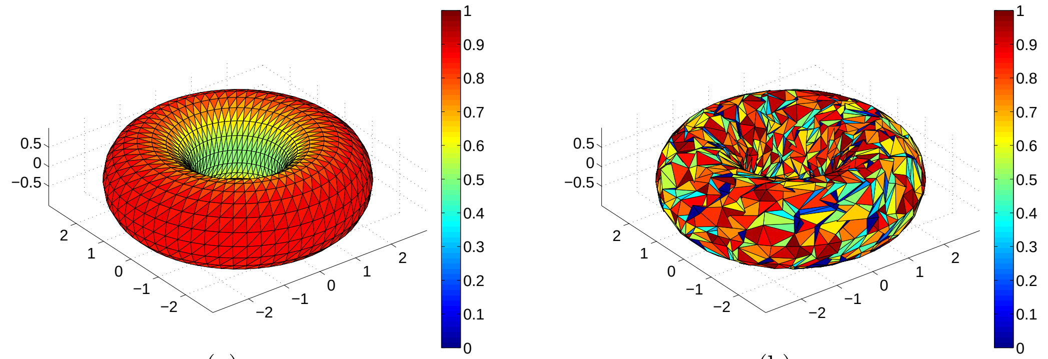
In the triangular case we extract this information for each one of the proposed numerical minimizing methods. From the different results of the methods we deduce which is the best one. For quadrilateral meshes we just analyse the computational aspects of the best minimization method for triangular meshes, since the minimization process for quadrilaterals is based on the algorithm for triangles. In the quadrilateral case we focus on the comparison between the two possible objective functions. The computational information of both functions and the quality statistics obtained with each of them is presented for all the examples. Looking at the presented results, conclusions are drawn on whether the reduced objective function (2.16) can be used instead of (2.15) or not.

5.1 Examples of triangular element meshes

In this section we apply the minimization process presented in Chapter 4 to four meshes composed by triangular elements that have been generated on parametric surfaces.

5.1.1 Example 1: Torus

Figure 5.1(a) presents an initial mesh created on a torus. This mesh is composed by 1200 nodes and 2242 elements. Figure 5.1(b) presents a mesh with the same topology than 5.1(a) but with almost half of the elements tangled. Starting from the tangled mesh, we have applied the smoothing-untangling procedure for surfaces and we have obtained the smoothed mesh presented in Figure 5.2.



(a) (b)

Figure 5.1: Surface meshes for a torus: (a) initial mesh, and (b) tangled mesh.

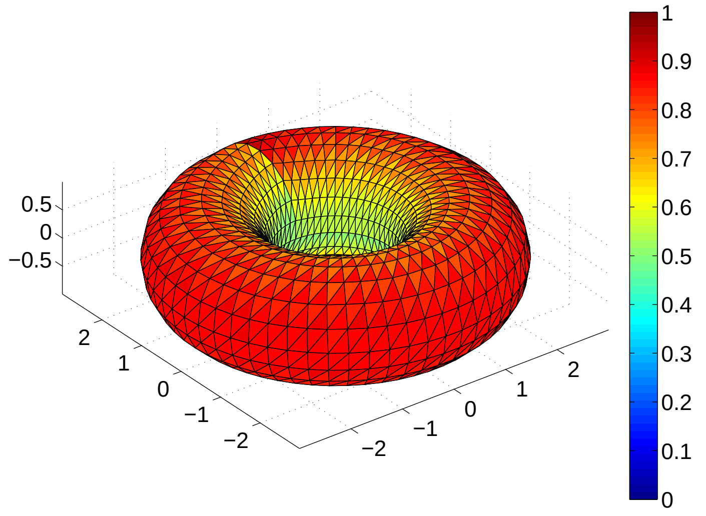


Figure 5.2: Smoothed and untangled surface mesh for a torus.

Table 5.1 presents statistical information corresponding to the meshes presented in this example. It is important to point out that the proposed algorithm has been able to untangle an input mesh with 545 tangled elements.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tangled el.**
Initial	0.50	0.87	0.73	0.13	0
Tangled	0.00	0.99	0.47	0.34	545
Smoothed	0.50	0.93	0.74	0.11	0

Table 5.1: Quality statistics for meshes presented in Figures 5.1(a), 5.1(b) and 5.2.

Although the minimum quality of the smoothed mesh is equal to the one of the initial mesh for the torus, the maximum and mean qualities have been increased. In addition, the standard deviation of the smoothed mesh is smaller that the initial one.

5.1.2 Example 2: Cup

Figure 5.3(a) presents an initial mesh created on a model of a cup. This mesh is composed by 675 nodes and 1232 elements. Figure 5.3(b) presents a tangled mesh on the cup with the same topology than 5.3(a). Taking as input the tangled mesh we have obtained the smoothed mesh is presented in Figure 5.4.

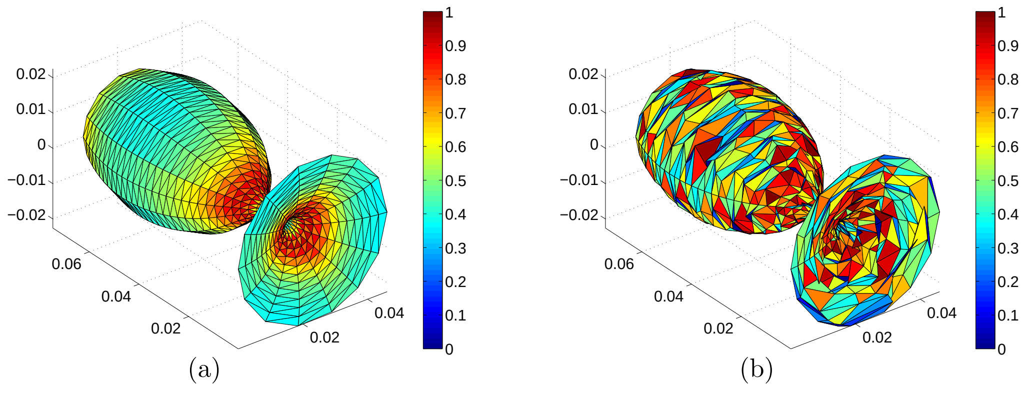


Figure 5.3: Surface meshes for a cup: (a) initial mesh, and (b) tangled mesh.

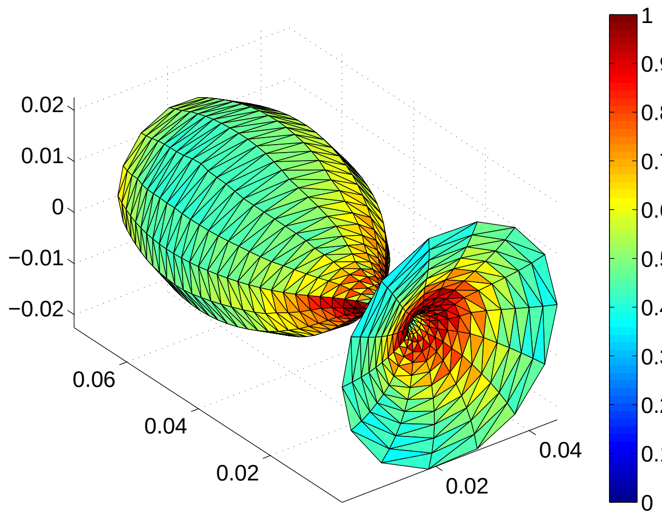


Figure 5.4: Smoothed and untangled surface mesh for a cup.

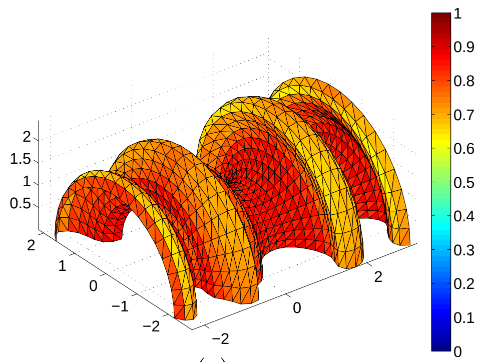
Table 5.2 presents statistical information of the meshes presented in Figures 5.3 and 5.4. It is important to point out that the proposed algorithm has been able to untangle an input mesh with 249 tangled elements. Note that in this example all the quality statistics presented are improved with respect with the initial mesh created on the presented geometry. Recall that the minimum quality over all the elements of the mesh has increased from 0.30 to 0.35. The standard deviation is equal in both the initial and the smoothed meshes.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tangled el.**
Initial	0.30	0.93	0.55	0.15	0
Tangled	0.00	0.99	0.42	0.29	249
Smoothed	0.35	0.99	0.58	0.15	0

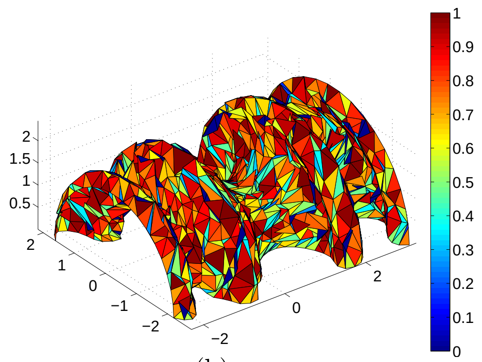
Table 5.2: Quality statistics for meshes presented in Figures 5.3(a), 5.3(b) and 5.4.

5.1.3 Example 3: Undulated surface

Figure 5.5(a) presents an initial mesh created on an undulated surface generated from the revolution of a curve. This mesh is composed by 1200 nodes and 2242 elements. Figure 5.5(b) presents a mesh with the same topology than 5.5(a) but with almost all the nodes defining tangled submeshes. Using the presented algorithm to smooth and untangle 5.5(b) we obtain the mesh presented in Figure 5.6.



(a)



(b)

Figure 5.5: Surface meshes for an undulated surface: (a) initial mesh, and (b) tangled mesh.

Table 5.3 presents statistical information of the different meshes presented in this example. Recall that the proposed algorithm has been able to untangle an input mesh with a high number of tangled elements. Note that in this example all the qualities presented are improved with respect with the initial mesh created on

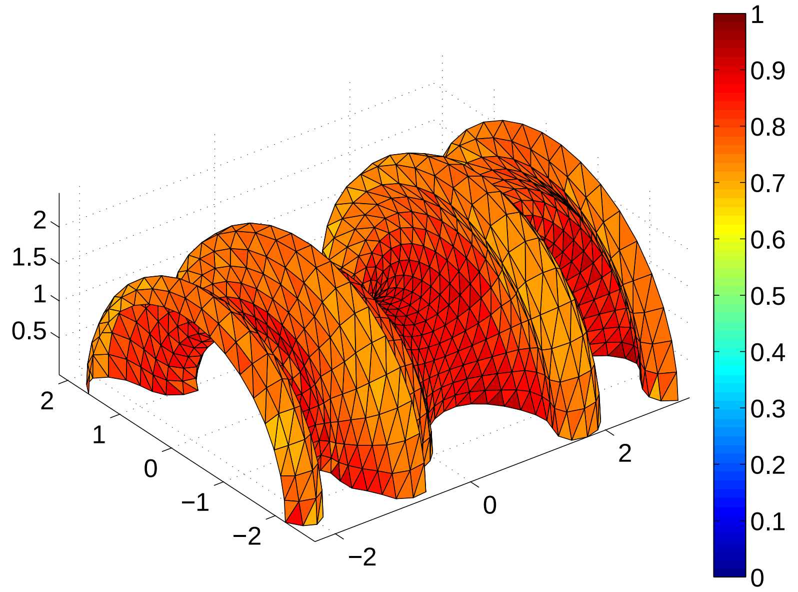


Figure 5.6: Smoothed and untangled surface mesh for an undulated surface.

the presented geometry. Once again the minimum is increased, from 0.62 to 0.67. Notice that an important improvement is presented on the maximum quality, that is incremented from 0.89 to 0.97. The standard deviation is also slightly reduced, obtaining a mesh with elements with a more uniform quality.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tangled el.**
Initial	0.62	0.89	0.79	0.06	0
Tangled	0.00	0.99	0.48	0.34	564
Smoothed	0.67	0.97	0.81	0.05	0

Table 5.3: Quality statistics for meshes presented in Figures 5.5(a), 5.5(b) and 5.6.

5.1.4 Example 4: Rolled surface

Figure 5.7(a) presents an initial mesh created on a rolled surface. This mesh is composed by 1800 nodes and 3332 elements. Figure 5.7(b) presents a tangled mesh on the surface conserving the topology of the initial mesh 5.7(a). Using the presented algorithm to smooth and untangle 5.7(b) we obtain the mesh presented in Figure 5.8. Note that Figure 5.8 contains two views of the smoothed mesh in order to facilitate the understanding of the presented geometry.

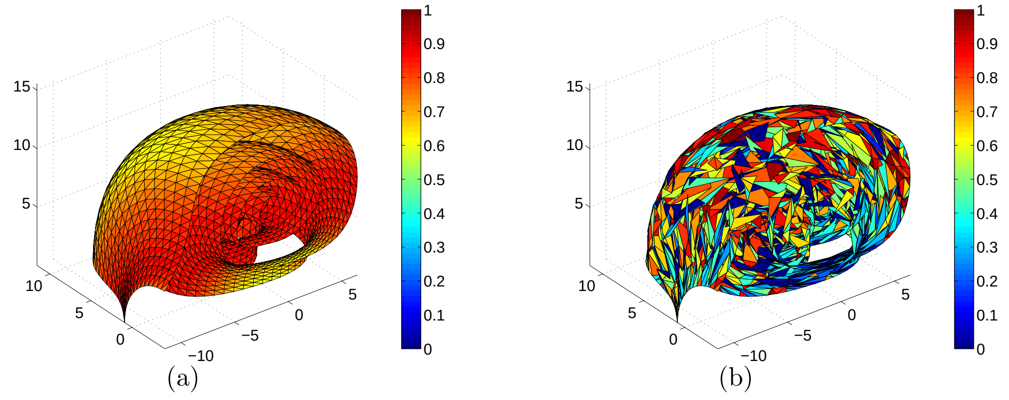


Figure 5.7: Surface meshes for a rolled surface: (a) initial mesh, and (b) tangled mesh.

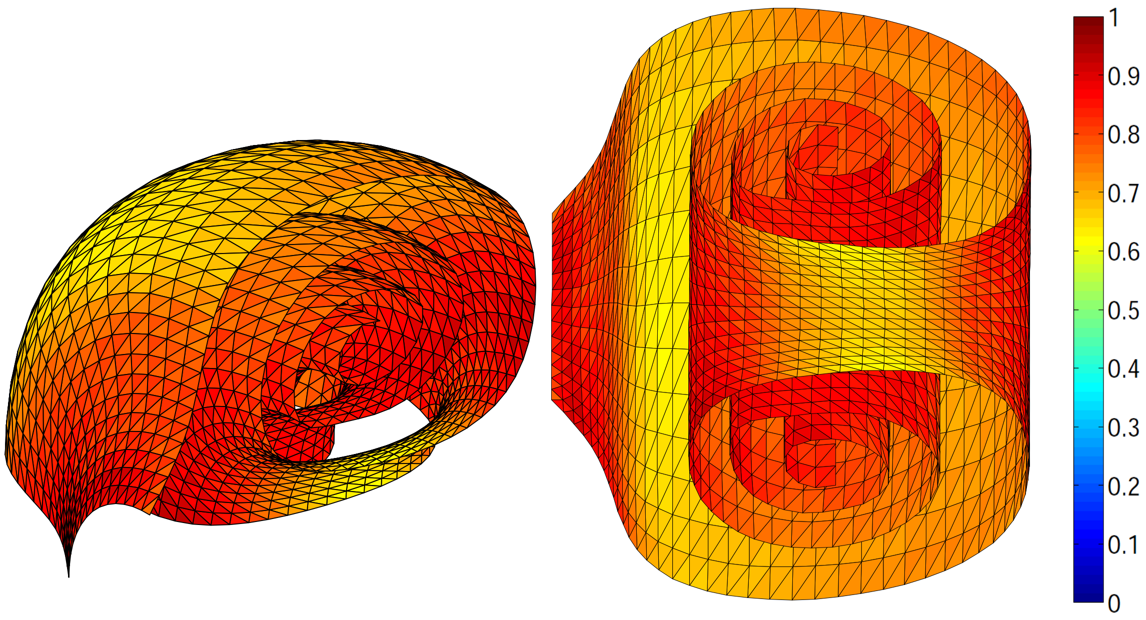


Figure 5.8: Two views of the smoothed and untangled surface mesh for a rolled surface.

Table 5.4 presents statistical information of the different meshes presented in Figures 5.7 and 5.8. Note that the proposed algorithm is able to untangle a mesh with 1191 tangled elements achieving better minimum, maximum and mean quality than in the initial presented mesh. The minimum is increased from 0.60 to 0.62 and the maximum quality is incremented from 0.88 to 0.97. The mean quality is slightly increased and standard deviation is conserved with respect to the initial mesh for the surface.

Mesh	Min. Q.	Max. Q.	Mean Q.	Std. Dev.	Tangled el.
Initial	0.60	0.88	0.78	0.07	0
Tangled	0.00	0.99	0.33	0.31	1191
Smoothed	0.62	0.97	0.79	0.07	0

Table 5.4: Quality statistics for meshes presented in Figures 5.7(a), 5.7(b) and 5.8.

5.2 Examples of quadrilateral element meshes

In this section we apply the minimization process presented in Chapter 4 to quadrilateral meshes. We use the same parametric surfaces that have been studied for triangle meshes. First, a structured quadrilateral mesh is created on the surface. Second, we tangle the initial mesh in order to check the capabilities of the presented theory for quadrilaterals. The tangled mesh is the input of the smoothing-untangling algorithm. Then, two different figures of the smoothed quadrilateral mesh are shown for each example. In the first one, we show the smoothed mesh obtained using objective function (2.15) for the optimization procedure. In the second figure we show the results when using objective function (2.16). The aim of the examples is to check that the presented theory for quadrilaterals is able to untangle and smooth quadrilateral meshes obtaining good quality final meshes.

5.2.1 Example 1: Torus

Figure 5.9(a) presents an initial mesh created on a torus. This mesh is composed by 1200 nodes and 1121 elements. Figure 5.9(b) presents a tangled mesh on the torus conserving the topology of 5.9(a). Taking as input the tangled mesh and applying the presented algorithm with the two objective functions for triangles, we have obtained the untangled meshes presented in Figure 5.10.

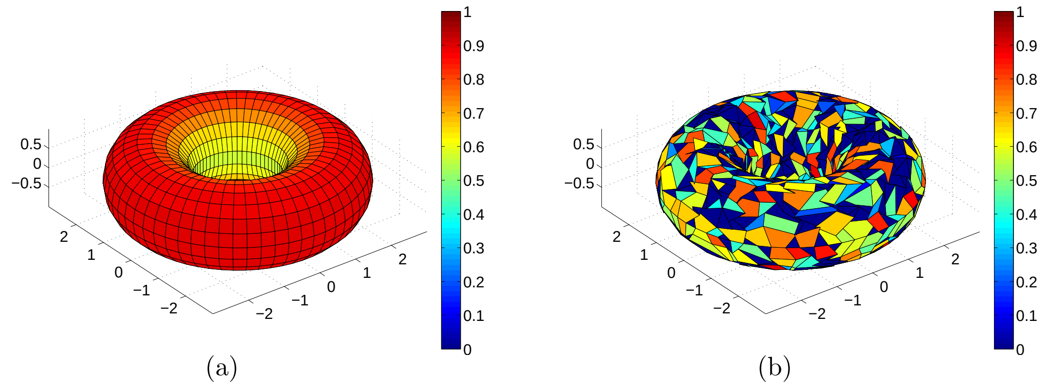


Figure 5.9: Surface meshes for a torus: (a) initial mesh, and (b) tangled mesh.

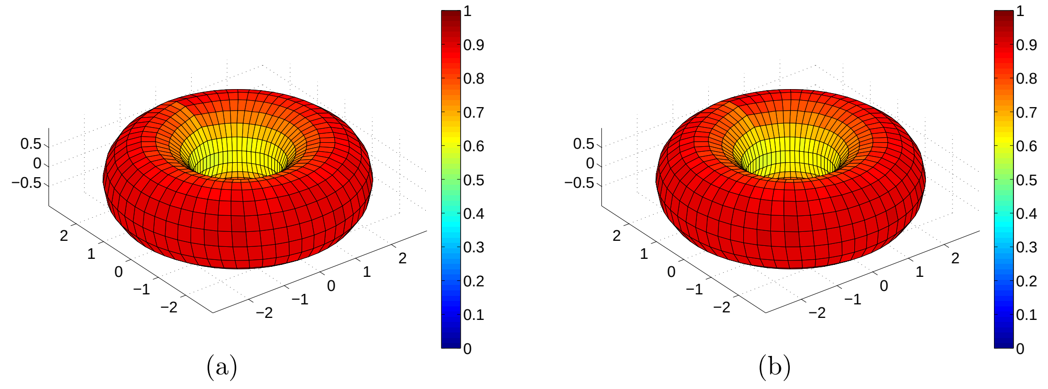


Figure 5.10: Smoothed surface mesh for a torus using: (a) objective function (2.15), and (b) objective function (2.16).

Table 5.5 presents statistical information of the meshes presented in Figures 5.9 and 5.10. It is important to point out that the proposed algorithm has been able to untangle an input mesh with 568 tangled elements. Note that in this example all the quality statistics presented are improved with respect to the initial mesh created on the presented geometry, independently of the objective function that we select.

Mesh	Min. Q.	Max. Q.	Mean Q.	Std. Dev.	Tang. el.
Initial	0.57	0.89	0.78	0.11	0
Tangled	0.00	0.92	0.27	0.30	568
Smoothed 5.10(a)	0.58	0.91	0.80	0.10	0
Smoothed 5.10(b)	0.58	0.92	0.80	0.09	0

Table 5.5: Quality statistics for meshes presented in Figures 5.9 and 5.10.

5.2.2 Example 2: Cup

Figure 5.11(a) presents an initial mesh created on a cup. This mesh is composed by 675 nodes and 616 elements. Figure 5.11(b) presents a mesh with the same topology than 5.11(a) but with almost half of the elements tangled. Starting from the tangled mesh, we have applied the smoothing-untangling procedure for surfaces and we have obtained the two smoothed meshes presented in Figure 5.12.

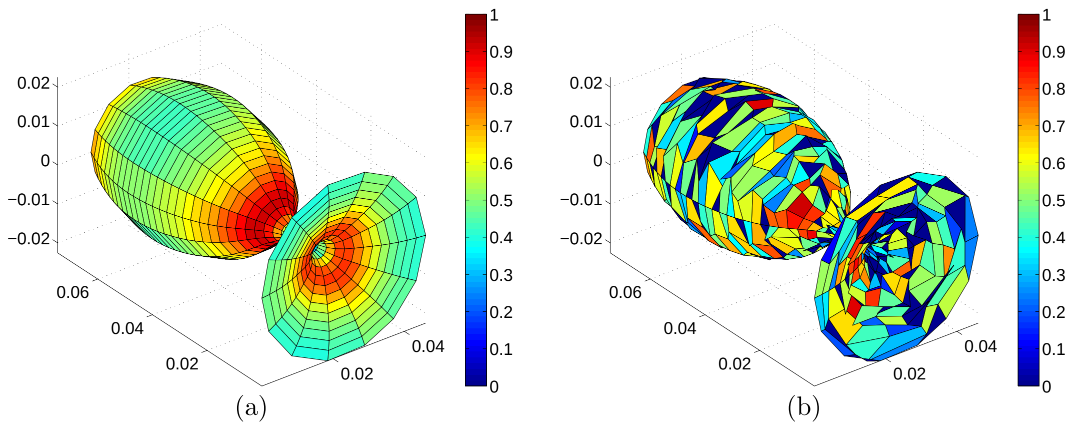


Figure 5.11: Surface meshes for a cup: (a) initial mesh, and (b) tangled mesh.

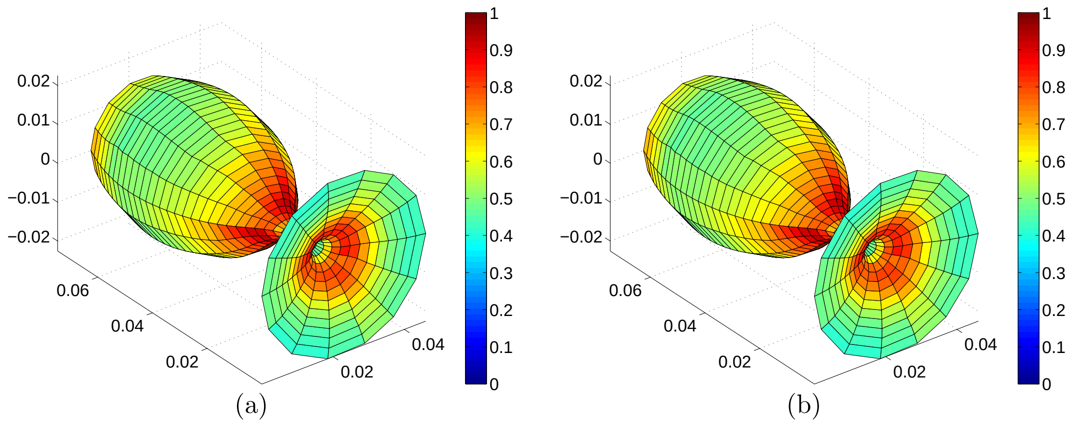


Figure 5.12: Smoothed surface mesh for a cup using: (a) objective function (2.15), and (b) objective function (2.16).

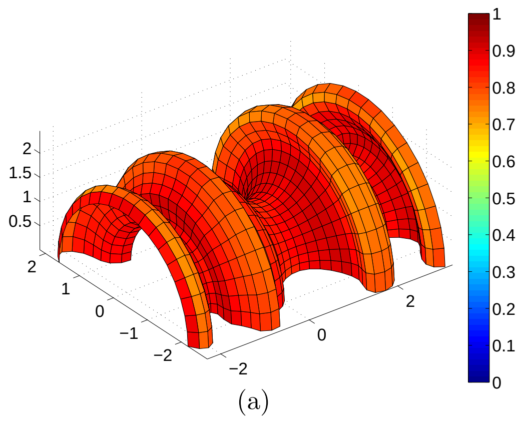
Table 5.6 presents statistical information of the different meshes presented in this example. Recall that the proposed algorithm has been able to untangle an input mesh with a high number of tangled elements. Note that in this example all the qualities presented are improved with respect with the initial mesh created on the presented geometry. Note that both objective functions present a final mesh with almost the same quality.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tang. el.**
Initial	0.29	0.91	0.59	0.15	0
Tangled	0.00	0.90	0.29	0.27	236
Smoothed 5.12(a)	0.32	0.93	0.61	0.14	0
Smoothed 5.12(b)	0.33	0.94	0.61	0.14	0

Table 5.6: Quality statistics for meshes presented in Figures 5.11 and 5.12.

5.2.3 Example 3: Undulated surface

Figure 5.13(a) presents an initial mesh created on a undulated surface. This mesh is composed by 1200 nodes and 1121 elements. Figure 5.13(b) presents a tangled mesh on the surface conserving the topology of the initial mesh 5.13(a). Figures 5.14(a) and 5.14(b) show the smoothed and untangled meshes by means of objective functions (2.15) and (2.16), respectively.


(a)

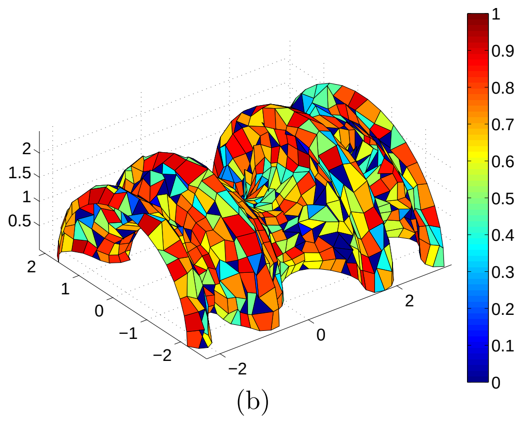

(b)

Figure 5.13: Surface meshes for an undulated surface: (a) initial mesh, and (b) tangled mesh.

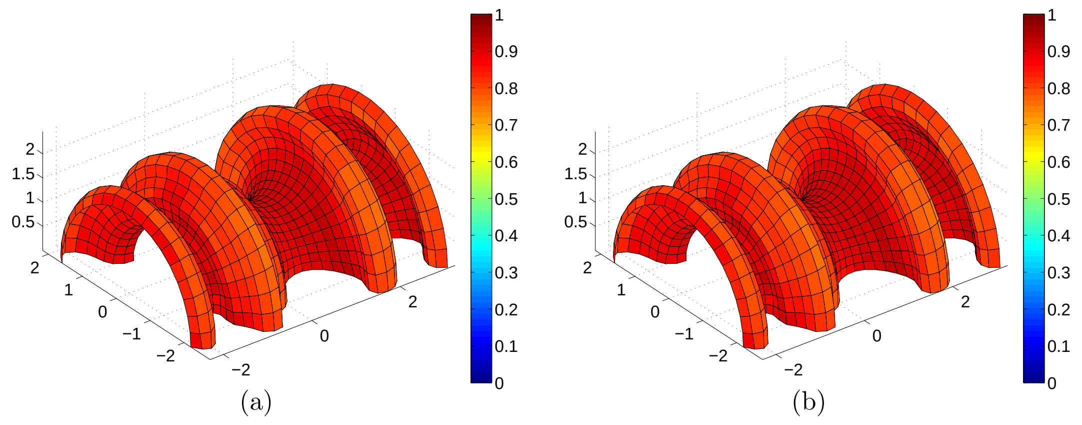


Figure 5.14: Smoothed surface mesh for an undulated surface using: (a) objective function (2.15), and (b) objective function (2.16).

Table 5.7 presents statistical information of the meshes presented in Figures 5.13 and 5.14. It is important to point out that the proposed algorithm has been able to untangle an input mesh with 264 tangled elements. Note that in this example all the quality statistics presented are improved with respect to the initial mesh created on the presented geometry, using objective function (2.15) or either using (2.16). Note that both (2.15) and (2.16) present no substantial differences in the quality statistics.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tang. el.**
Initial	0.62	0.89	0.79	0.06	0
Tangled	0.00	0.99	0.48	0.34	264
Smoothed 5.14(a)	0.71	0.93	0.84	0.05	0
Smoothed 5.14(b)	0.71	0.93	0.86	0.04	0

Table 5.7: Quality statistics for meshes presented in Figures 5.13 and 5.14.

5.2.4 Example 4: Rolled surface

Figure 5.15(a) presents an initial mesh on rolled surface. This mesh is composed by 1800 nodes and 1666 elements. Figure 5.15(b) presents a mesh with the same topology than 5.15(a) but with almost all the nodes defining tangled submeshes. Using the presented algorithm to smooth and untangle 5.15(b) we obtain the meshes presented in Figure 5.16.

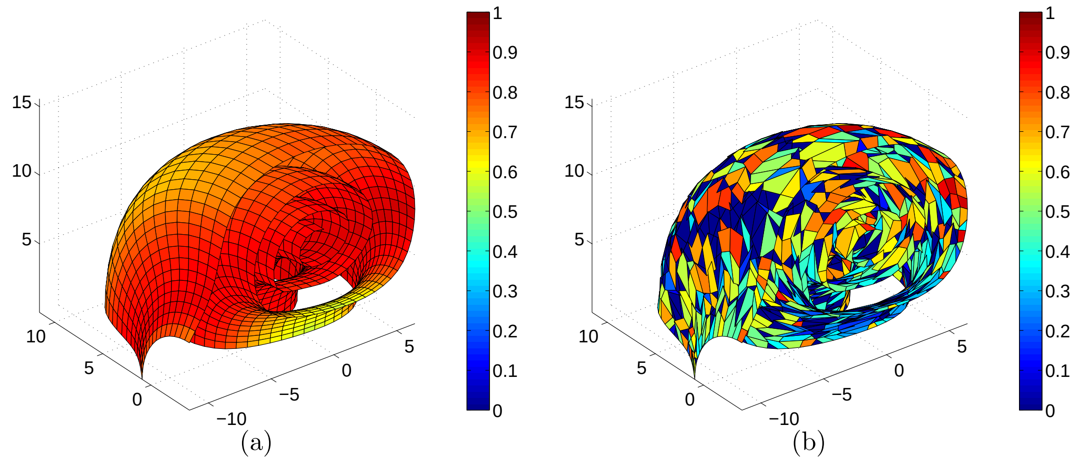


Figure 5.15: Surface meshes for a rolled surface: (a) initial mesh, and (b) tangled mesh.

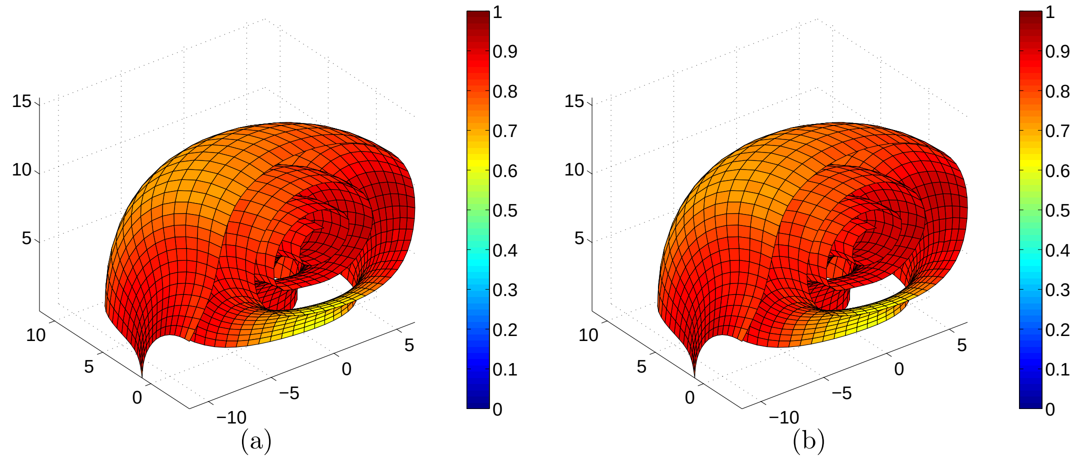


Figure 5.16: Smoothed surface mesh for a rolled surface using: (a) objective function (2.15), and (b) objective function (2.16).

Table 5.8 presents statistical information corresponding to the meshes presented in this example. It is important to point out that the proposed algorithm has been able to untangle an input mesh with 457 tangled elements. Note that smoothing with both objective functions all the quality aspects of the final meshes improve respect to the ones of the initial one. Comparing the results of (2.15) and (2.16) we realize that they are almost equal, but (2.16) presents slightly better results.

Mesh	**Min. Q.**	**Max. Q.**	**Mean Q.**	**Std. Dev.**	**Tang. el.**
Initial	0.56	0.90	0.82	0.07	0
Tangled	0.00	0.93	0.53	0.28	457
Smoothed 5.16(a)	0.59	0.92	0.83	0.07	0
Smoothed 5.16(b)	0.60	0.93	0.83	0.07	0

Table 5.8: Quality statistics for meshes presented in Figures 5.15 and 5.16.

5.3 Computational aspects of the presented examples

In this section we analyse the performance of the smoothing procedure for all the examples. For the triangle meshes we compare the performance of the four minimization methods proposed in Section 4.2. Based on the results of this comparison we fix the method with the best performance. Then, for quadrilateral surface meshes we analyse which of the two presented modifications, (2.15) or (2.16), of the standard objective function (2.10) carries out a better performance.

For the meshes composed by triangles we have used four numerical minimizing methods: the Newton-Raphson, the Steepest Descent, the BFGS and the modified Newton-Rapshon (line search Newton with Hessian modification). For the three line search methods we have used the Bactracking line search method presented in Algorithm 4.9 as the step length selection criterion.

For each one of the examples corresponding to triangular meshes, tables 5.9, 5.10, 5.11 and 5.12 present

- the number of iterations of the global Algorithm 4.3,
- the elapsed time required to smooth the whole mesh,
- the elapsed time per iteration, and
- the elapsed time per iteration and node

of each one of the four minimizing methods. Note that all the computational times are presented in seconds.

Statistics	N-R	Steepest D.	BFGS	Modified N-R
Number of iterations	21	68	36	11
Global elapsed time	3.86	24.78	138.07	3.59
Elapsed time per iteration	0.18	0.36	3.84	0.33
Time per iter. and node	$1.53{\cdot}10^{-4}$	$3.03{\cdot}10^{-4}$	$3.19{\cdot}10^{-3}$	$2.72{\cdot}10^{-4}$

Table 5.9: Computational aspects corresponding to example 1 in Section 5.1.1. Mesh composed by 1200 nodes and 2242 elements.

Statistics	N-R	Steepest D.	BFGS	Modified N-R
Number of iterations	123	38	49	8
Global elapsed time	14.85	96.03	1976.76	3.02
Elapsed time per iteration	0.12	2.53	40.34	0.38
Time per iter. and node	$1.78{\cdot}10^{-4}$	$3.74{\cdot}10^{-3}$	$5.97{\cdot}10^{-2}$	$5.59{\cdot}10^{-4}$

Table 5.10: Computational aspects corresponding to example 2 in Section 5.1.2. Mesh composed by 675 nodes and 1222 elements.

Statistics	N-R	Steepest D.	BFGS	Modified N-R
Number of iterations	28	60	60	6
Global elapsed time	3.94	367.01	3443.62	3.25
Elapsed time per iteration	0.14	6.12	57.39	0.54
Time per iter. and node	$1.17{\cdot}10^{-4}$	$5.09{\cdot}10^{-3}$	$4.78{\cdot}10^{-2}$	$4.51{\cdot}10^{-4}$

Table 5.11: Computational aspects corresponding to example 3 in Section 5.1.3. Mesh composed by 1200 nodes and 2242 elements.

Statistics	N-R	Steepest D.	BFGS	Modified N-R
Number of iterations	13	40	7	4
Global elapsed time	4.07	82.13	239.60	2.71
Elapsed time per iteration	0.31	2.05	34.23	0.68
Time per iter. and node	$9.78{\cdot}10^{-5}$	$6.41{\cdot}10^{-5}$	$1.06{\cdot}10^{-2}$	$2.11{\cdot}10^{-4}$

Table 5.12: Computational aspects corresponding to example 4 in Section 5.1.4. Mesh composed by 1800 nodes and 3332 elements.

It is important to point out that all the methods leads to the same smoothed mesh except in Example 3. In this example we have stopped the Steepest Descent and BFGS after 60 iterations because convergence was not achieved. Note that in this example the crests of the surface are discretized by just a few elements. Therefore, both minimizing methods are not able to compute the advancing direction with enough accuracy and hence do not converge to the final optimal configuration.

From Table 5.9 to Table 5.12 we realize that the modified Newton-Raphson is the fastest method in terms of global elapsed time. In addition, it is also the method that needs less number of iterations to converge. However, the elapsed time per iteration and the elapsed time per iteration and node of the modified Newton-Raphson is higher than the corresponding value of the standard Newton-Raphson method. Note that despite that standard Newton-Raphson has a smaller cost per iteration, it takes more iterations to converge and hence the global elapsed time is higher than for the modified version.

It is worth to notice that both the modified and the standard Newton-Raphson methods outperform the Steepest Descent and the BFGS methods. In particular, the elapsed time or the number of iterations for the Steepest Descent and the BFGS methods can be one or two orders of magnitude higher than the corresponding values for the modified and the standard Newton-Raphson.

From these results we conclude that the modified Newton-Raphson method is the best option for our purposes. Since the objective function for quadrilateral meshes is based on the objective functions for triangles we only will use the modified Newton-Raphson method to smooth and untangle quadrilateral surface meshes. However, in Section 2.5.1 we have introduced two objective functions for quadrilateral surface meshes, see equations (2.15) and (2.16). Therefore, in Table 5.13 we compare the computational efficiency of the modified Newton-Raphson method when it is applied to minimize these two objective functions. Note that all the presented computational times are in seconds, except the time per iteration and node, that is given in milliseconds. In addition, Table 5.13 also presents statistics about the quality of the meshes.

	Example 1		Example 2		Example 1		Example 4	
	(2.15)	(2.16)	(2.15)	(2.16)	(2.15)	(2.16)	(2.15)	(2.16)
Number it.	11	8	10	9	13	11	9	8
Elapsed time	3.97	2.60	4.77	3.17	6.64	4.59	9.15	6.41
Time per it.	0.36	0.32	0.48	0.35	0.51	0.42	1.02	0.80
Time/(it·node)	0.30	0.27	0.71	0.52	0.43	0.35	0.57	0.44
Min. Qual.	0.58	0.58	0.32	0.33	0.71	0.71	0.59	0.60
Max. Qual.	0.91	0.92	0.93	0.94	0.93	0.93	0.92	0.93
Mean Qual.	0.80	0.80	0.61	0.61	0.84	0.86	0.83	0.83
Std. dev.	0.10	0.09	0.14	0.14	0.05	0.04	0.07	0.07
Tang. el.	0	0	0	0	0	0	0	0

Table 5.13: Statistics of objectives function (2.15) and (2.16): computational and quality aspects of the presented examples.

Comparing the results obtained using objective functions (2.15) and (2.16) in Table 5.13 we conclude that objective function (2.16) provides better results in terms of number of iterations, elapsed time, elapsed time per iterations and elapsed time per iteration and node. It is important to point out that although objective function (2.16) reduces the number of triangles that are used to compute the objective function, it does not reduce the robustness of the approach. Moreover, notice that using objective function (2.16) the final quality of the mesh is always equal or better than with objective function (2.15). Hence, the reduced objective function for quadrilaterals (2.16) has proved computationally more efficient without penalizing the quality of the final mesh.

Chapter 6

Conclusions and future research

In this work a new procedure to smooth and untangle meshes on parameterized surfaces has been developed. This procedure is based on a numerical minimization of an objective function locally defined on the parametric coordinates of each node of the surface mesh. It is important to point out that the proposed procedure is valid for triangular and quadrilateral surface meshes. Moreover, for quadrilateral elements we propose a new objective function that requires less computational effort than the standard one.

The minimization process is performed in the parametric space. Specifically, we propose an extension of any planar element quality measure to elements on parameterized surfaces. This prevents the requirement of smoothing the surface submeshes through an approximation in a certain tangent plane.

To improve the robustness of the minimization process, we have proposed two modifications of the objective function to improve the robustness of the minimization process. These modifications are based on a standard technique and are controlled by a scalar parameter. In the first modification a different scalar parameter is computed on each node of the surface mesh. In the second one, a fixed value of the scalar parameter is determined for a reference size. Then, all the elements are scaled to the reference size and the fixed value of the scalar parameter is used on each node of the surface mesh.

Several minimization algorithms have been implemented and compared both on triangular and quadrilateral surface meshes. From our experimental results we conclude that the modified Newton-Raphson (line search Newton with Hessian modification) outperforms the standard Newton-Raphson method and the other analysed line-search methods both in terms of the total time and the time per iteration.

An important contribution of this project is the improvement of the computational performance of the minimization process. In particular, we first group the nodes with the same number of adjacent elements. Second, in each one of these groups we colour the nodes in independent sets (two adjacent nodes have different colours). Then, the coordinates of the nodes in each colour set can be stored in a static array and their location can be optimized at the same time. This results in a vectorized and fast implementation of the method.

The results of this work lead to a new scientific and technological challenges. For instance,

- The parallelization of the current code. Since we have proposed a vectorized implementation, where the nodes in the same colour are independent, we propose to implement the code in parallelized manner. Specifically, we propose to move the nodes with the same colour in parallel. We consider to implement this technique in a vector processor such as graphics processing units (GPU).
- The implementation of the proposed method in a mesh generation environment *EZ4U*. Therefore, the MATLAB (interpreted) code will be translated to C++ (compiled). This will improve the efficiency of the method and will enlarge its applicability.
- The extension of the developed ideas for high-order elements. In the near future we would like to develop a quality measure for high order elements, and then extend it to surface problems.

Appendix A

Derivatives of the parametric objective function

A.1 First order derivatives of the parametric objective function

In order to compute the derivative of the parametric objective function, we use the chain rule in the main expression

$$\frac{\mathrm{d}\hat{\eta}_3}{\mathrm{d}\mathbf{u}}(\mathbf{u}) = \frac{\mathrm{d}(\hat{\eta}_2 \circ \lambda \circ \mathbf{T} \circ \varphi)}{\mathrm{d}\mathbf{u}}(\mathbf{u}) = D\hat{\eta}_2|_{\lambda(\mathbf{T}(\varphi(\mathbf{u})))} \cdot D\lambda|_{\mathbf{T}(\varphi(\mathbf{u}))} \cdot D\mathbf{T}|_{\varphi(\mathbf{u})} \cdot \frac{\mathrm{d}\varphi}{\mathrm{d}\mathbf{u}}|_{\mathbf{u}}. \quad (A.1)$$

In this Appendix we compute the different derivatives that appear in the chain rule that we have just shown.

A.1.1 Derivatives of φ

The derivative of the parametrization that we use will depend on the surface we are working with in each case, or in the area of the surface we are in. Thus, we directly give to the optimization procedure the analytical expression of such derivatives.

A.1.2 Derivatives of T

In these appendices we will compute the derivatives of **T**, that is the new component of our construction. We write **T** function as

$$T(\mathbf{y}) = \mathbf{M}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1),$$

and thus,

$$\frac{\partial T(\mathbf{y})}{\partial \alpha} = \frac{\partial \mathbf{M}}{\partial \alpha}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1) + \mathbf{M}(\mathbf{y}) \cdot \mathbb{I}_\alpha,$$

where $\mathbb{I}_\alpha$ is the zero vector with a 1 value on coordinate α.

In the computation of the second derivatives of $\mathbf{M}$, we will introduce the notation $\mathbf{e}_2^{\text{prev}} = \mathbf{e}_2 - (\mathbf{e}_2^T \cdot \tilde{\mathbf{e}}_1)\tilde{\mathbf{e}}_1$, in order to have more compact expressions. Lets proceed with the computation of the derivative of the matrix $\mathbf{M}$:

$$
\begin{aligned}
\frac{\partial \mathbf{M}}{\partial \alpha}(\mathbf{y}) &= \begin{pmatrix} \mathbf{0} \\ \frac{\partial \tilde{\mathbf{e}}_2^T}{\partial \alpha}(\mathbf{y}) \end{pmatrix}, \\
\frac{\partial \tilde{\mathbf{e}}_2}{\partial \alpha}(\mathbf{y}) &= \frac{\partial}{\partial \alpha}\left(\text{sign} \cdot \tilde{\mathbf{e}}_{2,0}\right) = \frac{\partial \text{sign}}{\partial \alpha} \cdot \tilde{\mathbf{e}}_{2,0} + \text{sign} \cdot \frac{\partial}{\partial \alpha}\tilde{\mathbf{e}}_{2,0}, \\
\frac{\partial \text{sign}}{\partial \alpha} &= \frac{\partial}{\partial \alpha} \frac{\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|} \\
&= \frac{\partial \det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}}{\partial \alpha} \cdot \frac{1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|} + \det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})} \cdot \frac{\partial}{\partial \alpha} \frac{1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|}, \\
\frac{\partial}{\partial \alpha} \frac{1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|} &= -\frac{1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|^2} \frac{\partial}{\partial \alpha} |\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}| \\
&= -\frac{1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|^2} \cdot (\pm 1) \cdot \frac{\partial \det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}}{\partial \alpha} \\
&= -\frac{\pm 1}{|\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}|^2} \cdot \frac{\partial \det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}}{\partial \alpha} \\
\frac{\partial}{\partial \alpha} \det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})} &= \frac{\partial}{\partial \alpha}\left((\mathbf{n} \times \tilde{\mathbf{e}}_1) \cdot \tilde{\mathbf{e}}_{2,0}\right) = (\mathbf{n} \times \tilde{\mathbf{e}}_1) \cdot \frac{\partial}{\partial \alpha} \tilde{\mathbf{e}}_{2,0}, \\
\frac{\partial}{\partial \alpha} \tilde{\mathbf{e}}_{2,0} &= \frac{\partial}{\partial \alpha} \frac{\mathbf{e}_2^{\text{prev}}}{\|\mathbf{e}_2^{\text{prev}}\|} = \\
&= \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|} \frac{\partial \mathbf{e}_2^{\text{prev}}}{\partial \alpha} + \mathbf{e}_2^{\text{prev}} \frac{\partial}{\partial \alpha} \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|}, \\
\frac{\partial}{\partial \alpha} \|\mathbf{e}_2^{\text{prev}}\|^{-1} &= -\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|^2} \frac{\partial}{\partial \alpha} \|\mathbf{e}_2^{\text{prev}}\| \\
&= -\frac{1}{2} \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|^3} \cdot \frac{\partial}{\partial \alpha}((\mathbf{e}_2^{\text{prev}})^T (\mathbf{e}_2^{\text{prev}})) \\
&= -\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|^3} \left((\mathbf{e}_2^{\text{prev}})^T \cdot \frac{\partial \mathbf{e}_2^{\text{prev}}}{\partial \alpha} \right) \\
\frac{\partial \mathbf{e}_2^{\text{prev}}}{\partial \alpha} &= \mathbb{I}_\alpha - (\tilde{\mathbf{e}}_1 \cdot \mathbb{I}_\alpha)\tilde{\mathbf{e}}_1 = \mathbb{I}_\alpha - \tilde{\mathbf{e}}_1^\alpha \tilde{\mathbf{e}}_1
\end{aligned}
$$

Note that in order to simplify the notation we have avoided to explicit the terms that do depend on $\mathbf{y}$, and in order to reduce some unnecessary space we have written $\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})}$ instead of $\det(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})$. Moreover, we have to point out that the sign function is indeed not derivable everywhere. Since it contains the absolute value of the determinant of $\tilde{\mathbf{e}}_1$, $\tilde{\mathbf{e}}_{2,0}$ and $\mathbf{n}$, sign is not derivable when $\det_{(\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_{2,0}, \mathbf{n})} = 0$.

A.1.3 Derivatives of λ

Recall that $\lambda(\mathbf{x})$ has been defined in Equation (3.24) as $\lambda(\mathbf{x}) = \frac{1}{\Lambda} \cdot \mathbf{x}$. Note then that the Jacobian is the constant matrix

$$D\lambda = \frac{1}{\Lambda}\mathbf{I}_{2,2}, \tag{A.2}$$

the identity divided by the scalar factor Λ.

A.1.4 Derivatives of $\hat{\eta_2}$

The objective function for a certain triangle with a free node $\mathbf{u}$ is

$$\hat{\eta_2}_k(\mathbf{x}) = \frac{|\mathbf{S}_k(\mathbf{x})|^2}{2h(\sigma_k(\mathbf{x}))} = \frac{|\mathbf{S}_k(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2)|^2}{2h(\sigma_k(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2))}, \tag{A.3}$$

being $\sigma_k(\mathbf{x}) = \det(\mathbf{S}_k(\mathbf{x}))$.

Moreover $\mathbf{S}_k(\mathbf{x})$ was defined in (2.5) by

$$\mathbf{S}(\mathbf{x}) \equiv \mathbf{S}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2) = \mathbf{A}(\mathbf{x},\mathbf{x}_1,\mathbf{x}_2)\cdot\mathbf{W}^{-1}.$$

Then, to compute the derivatives of the objective function we must apply the chain rule to these expressions. In order to simplify the notation, lets denote by α the variable respect to which we are computing the derivative:

- First, we deduce the expression corresponding to the derivative of equation (A.3). Recall that $|\mathbf{S}| = \sqrt{(\mathbf{S},\mathbf{S})}$, where $(\mathbf{A},\mathbf{B}) = \mathrm{tr}(\mathbf{A}^T\mathbf{B})$. To this end, we also need the following derivatives:

  □ Derivative of the Frobenius norm of matrix $\mathbf{S}$:

  $$\frac{\partial|\mathbf{S}(\mathbf{x})|}{\partial\alpha} = \frac{\partial\sqrt{(\mathbf{S},\mathbf{S})}}{\partial\alpha} = \frac{1}{2}\frac{1}{\sqrt{(\mathbf{S},\mathbf{S})}}\frac{\partial(\mathbf{S},\mathbf{S})}{\partial\alpha} = \frac{1}{2}\frac{1}{|\mathbf{S}|}\frac{\partial(\mathbf{S},\mathbf{S})}{\partial\alpha} = \frac{\left(\frac{\partial\mathbf{S}}{\partial\alpha},\mathbf{S}\right)}{|\mathbf{S}|}.$$

  □ Derivative of function $h(\sigma)$ (see Equation (2.17)):

  $$\frac{\partial h(\sigma(\mathbf{x}))}{\partial\alpha} = \frac{\partial}{\partial\alpha}\left(\frac{1}{2}\left(\sigma+\sqrt{\sigma^2+4\delta^2}\right)\right) = \frac{1}{2}\left(1+\frac{\sigma}{\sqrt{\sigma^2+4\delta^2}}\right)\frac{\partial\sigma}{\partial\alpha}.$$

  □ To compute the derivative of σ we just have to apply the formula for the derivative of a determinant of a matrix, see reference [Golberg]. Then,

  $$\frac{\partial\sigma(\mathbf{x})}{\partial\alpha} = \frac{\partial\det(\mathbf{S})}{\partial\alpha} = \det(\mathbf{S})\mathrm{tr}\left(\mathbf{S}^{-1}\frac{\partial\mathbf{S}}{\partial\alpha}\right) = \sigma\mathrm{tr}\left(\mathbf{S}^{-1}\frac{\partial\mathbf{S}}{\partial\alpha}\right).$$

With these previous calculations, now lets proceed with the derivative of $\hat{\eta_2}_k(\mathbf{x})$:

$$
\begin{aligned}
\frac{\partial \hat{\eta_2}}{\partial \alpha}(\mathbf{x}) &= \frac{\partial |\mathbf{S}|^2}{\partial \alpha}\frac{1}{2h(\sigma)} + |\mathbf{S}|^2 \frac{\partial}{\partial \alpha}\left(\frac{1}{2h(\sigma)}\right) \\
&= \frac{1}{2h(\sigma)} 2|\mathbf{S}|\frac{\partial |\mathbf{S}|}{\partial \alpha} - |\mathbf{S}|^2 \frac{1}{2(h(\sigma))^2}\frac{\partial h(\sigma)}{\partial \alpha} \\
&= \frac{\left(\frac{\partial \mathbf{S}}{\partial \alpha}, \mathbf{S}\right)}{h(\sigma)} - \frac{|\mathbf{S}|^2}{2(h(\sigma))^2}\left(\frac{1}{2} + \frac{\sigma}{2\sqrt{\sigma^2+4\delta^2}}\right)\frac{\partial \sigma}{\partial \alpha} \\
&= 2\hat{\eta_2}\frac{\left(\frac{\partial \mathbf{S}}{\partial \alpha}, \mathbf{S}\right)}{|\mathbf{S}|^2} - \frac{\hat{\eta_2}}{h(\sigma)}\frac{\sigma + \sqrt{\sigma^2+4\delta^2}}{2\sqrt{\sigma^2+4\delta^2}}\frac{\partial \sigma}{\partial \alpha} \\
&= 2\hat{\eta_2}\frac{\left(\frac{\partial \mathbf{S}}{\partial \alpha}, \mathbf{S}\right)}{|\mathbf{S}|^2} - \frac{\hat{\eta_2}}{h(\sigma)}\frac{h(\sigma)}{\sqrt{\sigma^2+4\delta^2}}\frac{\partial \sigma}{\partial \alpha} \\
&= 2\hat{\eta_2}\left(\frac{\left(\frac{\partial \mathbf{S}}{\partial \alpha}, \mathbf{S}\right)}{|\mathbf{S}|^2} - \frac{\frac{\partial \sigma}{\partial \alpha}}{2\sqrt{\sigma^2+4\delta^2}}\right).
\end{aligned}
$$

- Second we deduce the expression corresponding to the partial derivative of the shape matrix $\mathbf{S}$:

$$\frac{\partial \mathbf{S}}{\partial \alpha}(\mathbf{x}) = \frac{\partial \mathbf{A}}{\partial \alpha}(\mathbf{x})\mathbf{W}^{-1}, \tag{A.4}$$

where $\mathbf{A}$ is defined in (2.3) as

$$\mathbf{A}(\mathbf{x}) = \mathbf{A}((x,y)) = \mathbf{A}((x,y),\mathbf{x}_1,\mathbf{x}_2) = \begin{pmatrix} x_1 - x & x_2 - x \\ y_1 - y & y_2 - y \end{pmatrix}. \tag{A.5}$$

Then, the partial derivatives are:

$$\frac{\partial \mathbf{A}}{\partial x}(x,y) = \begin{pmatrix} -1 & -1 \\ 0 & 0 \end{pmatrix}, \tag{A.6}$$

$$\frac{\partial \mathbf{A}}{\partial y}(x,y) = \begin{pmatrix} 0 & 0 \\ -1 & -1 \end{pmatrix}. \tag{A.7}$$

A.2 Second order derivatives of the objective function for a submesh

In the minimization procedure that we develop in order to find the optimal configuration of the mesh we need to compute the derivatives of the objective function. According (3.17), the objective function that we have used for the surface mesh optimization is

$$K_{\hat{\eta_3}}(\mathbf{u}) = \left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^p(\mathbf{u})\right)^{1/p}, \tag{A.8}$$

where $\hat{\eta_3}_k$ is defined in (3.13).

Then, to compute the derivatives of the objective function we must apply the chain rule to these expressions. In order to simplify the notation, lets denote by α the variable respect to which we are computing the derivative (since $\mathbf{u} = (u, v)^T$, then α can be equal to u or v).

The expression corresponding to the derivative of equation (A.8) is:

$$\begin{aligned}\frac{\partial K_{\hat{\eta_3}(\mathbf{u})}}{\partial\alpha} &= \frac{1}{p}\left(\sum_{k=1}^{m}(\hat{\eta_3}_k)^p(\mathbf{u})\right)^{\frac{1}{p}-1}\cdot\sum_{k=1}^{m}\left(p\,(\hat{\eta_3}_k)^{p-1}(\mathbf{u})\cdot\frac{\partial\hat{\eta_3}_k}{\partial\alpha}(\mathbf{u})\right)\\ &= \left(\sum_{k=1}^{m}(\hat{\eta_3}_k)^p(\mathbf{u})\right)^{\frac{1}{p}-1}\cdot\sum_{k=1}^{m}\left((\hat{\eta_3}_k)^{p-1}(\mathbf{u})\cdot\frac{\partial\hat{\eta_3}_k}{\partial\alpha}(\mathbf{u})\right).\end{aligned}$$

A.3 Computation of the second order derivatives of the parametric objective function

Recall the first order derivative of the parametric objective function, computed in Equation (A.1) in Appendix A.1:

$$\frac{\mathrm{d}\hat{\eta_3}}{\mathrm{d}\mathbf{u}}(\mathbf{u}) = \frac{\mathrm{d}(\hat{\eta_2}\circ\lambda\circ\mathbf{T}\circ\varphi)}{\mathrm{d}\mathbf{u}}(\mathbf{u}) = D\hat{\eta_2}|_{\lambda(\mathbf{T}(\varphi(\mathbf{u})))}\cdot D\lambda|_{\mathbf{T}(\varphi(\mathbf{u}))}\cdot D\mathbf{T}|_{\varphi(\mathbf{u})}\cdot\frac{\mathrm{d}\varphi}{\mathrm{d}\mathbf{u}}|_{\mathbf{u}}.$$

Note that , the Hessian of a composition of functions, is not in general the product of the Hessian matrices of the functions, as happens with the Jacobian matrices. Hence, in order to compute the second order derivative of the parametric objective function we require to use tensorial notation (and compute the derivative component by component). We first rewrite the matrix expression of the first derivative, avoiding to explicitly specify the point in which we evaluate each expression:

$$\begin{aligned}\frac{\partial\hat{\eta_3}}{\partial\alpha} &= \frac{\partial(\hat{\eta_2}\circ\lambda\circ\mathbf{T}\circ\varphi)}{\partial\alpha} = D\hat{\eta_2}\cdot D\lambda\cdot D\mathbf{T}\cdot\frac{\partial\varphi}{\partial\alpha}\\ &= D\hat{\eta_2}\cdot\frac{1}{\Lambda}\mathbf{I_{2,2}}\cdot D\mathbf{T}\cdot\frac{\partial\varphi}{\partial\alpha} = \frac{1}{\Lambda}D\hat{\eta_2}\cdot D\mathbf{T}\cdot\frac{\partial\varphi}{\partial\alpha}\\ &= \frac{1}{\Lambda}\sum_{i=1,2,3}\sum_{j=1,2}\left(\hat{\eta_2}_{x_j}\cdot\mathbf{T}^j_{y_i}\cdot\varphi^i_\alpha\right),\end{aligned}$$

We denote by (x_1, x_2) the two variables of the plane where we map τ and (y_1, y_2, y_3) the variables of $\mathbb{R}^3$ where we express the surface. A more compact expression can be written using Einstein notation in order to simplify the second order derivative:

$$\frac{\partial\hat{\eta_3}}{\partial\alpha} = \frac{1}{\Lambda}\hat{\eta_2}_a\mathbf{T}^a_b\varphi^b_\alpha,$$

where $\mathbf{f}_b^a$ denotes the derivative respect the variable b of the component a of function $\mathbf{f}$. Hence, the second derivative of $\hat{\eta_3}$ using Einstein notation is shown on Equation (A.9):

$$\frac{\partial^2 \hat{\eta_3}}{\partial\alpha\partial\beta} = \frac{1}{\Lambda}\left(\hat{\eta_2}_{ac}\mathbf{T}_d^c\varphi_\beta^b \cdot \mathbf{T}_b^a\varphi_\alpha^b + \hat{\eta_2}_a\mathbf{T}_{bc}^a\varphi_\beta^c \cdot \varphi_\alpha^b + \hat{\eta_2}_a\mathbf{T}_b^a\varphi_{\alpha\beta}^b\right). \tag{A.9}$$

Therefore, we need to compute the second derivatives of function φ, $\mathbf{T}$ and $\hat{\eta_2}$.

A.3.1 Second order derivatives of φ

As we commented on Appendix A.1.1, φ defines the surface in which the mesh is smoothed. Hence, for each surface the analytical expression of the derivatives is given.

A.3.2 Second order derivatives of T

In these appendices we will compute the second derivatives of $\mathbf{T}$. Recalling that

$$\begin{aligned} T(\mathbf{y}) &= \mathbf{M}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1), \text{ and} \\ \frac{\partial T(\mathbf{y})}{\partial\alpha} &= \frac{\partial \mathbf{M}}{\partial\alpha}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1) + \mathbf{M}(\mathbf{y}) \cdot \mathbb{I}_i, \end{aligned}$$

we continue with the expressions of the second order derivatives:

$$\begin{aligned} \frac{\partial^2 T(\mathbf{y})}{\partial\alpha\partial\beta} &= \frac{\partial}{\partial\beta}\left(\frac{\partial \mathbf{M}}{\partial\alpha}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1) + \mathbf{M}(\mathbf{y}) \cdot \mathbb{I}_\alpha\right) \\ &= \frac{\partial \mathbf{M}}{\partial\beta} \cdot \mathbb{I}_\alpha + \frac{\partial \mathbf{M}}{\partial\alpha} \cdot \mathbb{I}_\beta + \frac{\partial^2 \mathbf{M}}{\partial\alpha\partial\beta}(\mathbf{y}) \cdot (\mathbf{y} - \mathbf{y}_1). \end{aligned}$$

Note that in this expression there is just one component unknown: $\frac{\partial^2 \mathbf{M}}{\partial\alpha\partial\beta}(\mathbf{y})$, since the expressions related to the first derivatives have been previously computed.

Lets proceed with the computations of the second order derivatives of matrix $\mathbf{M}$:

$$\frac{\partial^2 \mathbf{M}}{\partial\alpha\partial\beta} = \frac{\partial^2}{\partial\alpha\partial\beta}\begin{pmatrix} \tilde{\mathbf{e}}_1^T \\ \tilde{\mathbf{e}}_2^T \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{\partial^2 \tilde{\mathbf{e}}_2^T}{\partial\alpha\partial\beta} \end{pmatrix},$$

$$\frac{\partial^2 \tilde{\mathbf{e}}_2}{\partial\alpha\partial\beta} = \frac{\partial}{\partial\beta}\left(\frac{\partial \tilde{\mathbf{e}}_{2,0}}{\partial\alpha}\cdot \text{sign} + \tilde{\mathbf{e}}_{2,0}\cdot\frac{\partial \text{sign}}{\partial\alpha}\right)$$

$$= \frac{\partial^2 \tilde{\mathbf{e}}_{2,0}}{\partial\alpha\partial\beta}\cdot\text{sign} + \frac{\partial \tilde{\mathbf{e}}_{2,0}}{\partial\alpha}\cdot\frac{\partial\text{sign}}{\partial\beta} + \frac{\partial \tilde{\mathbf{e}}_{2,0}}{\partial\beta}\cdot\frac{\partial\text{sign}}{\partial\alpha} + \tilde{\mathbf{e}}_{2,0}\cdot\frac{\partial^2\text{sign}}{\partial\alpha\partial\beta},$$

$$\frac{\partial^2\text{sign}}{\partial\alpha\partial\beta} = \frac{\partial}{\partial\beta}\left(\frac{\partial}{\partial\alpha}\det{}_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}\cdot\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|} + \det{}_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}\frac{\partial}{\partial\alpha}\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|}\right)$$

$$= \frac{\partial^2\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha\partial\beta}\cdot\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|} + \frac{\partial\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha}\frac{\partial}{\partial\beta}\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|}$$

$$+\frac{\partial\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha}\frac{\partial}{\partial\beta}\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|} + \det{}_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|},$$

$$\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|} = \frac{\partial}{\partial\beta}\left(-\frac{\pm 1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|^2}\cdot\frac{\partial\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha}\right)$$

$$= \frac{2}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|^3}\frac{\partial\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha}\cdot\frac{\partial|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|}{\partial\beta} - \frac{\pm 1}{|\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}|^2}\frac{\partial^2\det_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})}}{\partial\alpha\partial\beta},$$

$$\frac{\partial^2}{\partial\alpha\partial\beta}\det{}_{(\tilde{\mathbf{e}}_1,\tilde{\mathbf{e}}_{2,0},\mathbf{n})} = \frac{\partial^2}{\partial\alpha\partial\beta}\left((\mathbf{n}\times\tilde{\mathbf{e}}_1)\cdot\tilde{\mathbf{e}}_{2,0}\right) = (\mathbf{n}\times\tilde{\mathbf{e}}_1)\cdot\frac{\partial^2\tilde{\mathbf{e}}_{2,0}}{\partial\alpha\partial\beta},$$

$$\frac{\partial^2\tilde{\mathbf{e}}_{2,0}}{\partial\alpha\partial\beta} = \frac{\partial}{\partial\beta}\left(\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|}\cdot\frac{\partial}{\partial\alpha}\mathbf{e}_2^{\text{prev}} + \mathbf{e}_2^{\text{prev}}\cdot\frac{\partial}{\partial\alpha}\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|}\right)$$

$$= \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|}\cdot\frac{\partial^2}{\partial\alpha\partial\beta}\mathbf{e}_2^{\text{prev}} + \frac{\partial}{\partial\beta}\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|}\cdot\frac{\partial}{\partial\alpha}\mathbf{e}_2^{\text{prev}}$$

$$+\frac{\partial}{\partial\beta}\mathbf{e}_2^{\text{prev}}\cdot\frac{\partial}{\partial\alpha}\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|} + \mathbf{e}_2^{\text{prev}}\cdot\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|},$$

$$\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{\|\mathbf{e}_2^{\text{prev}}\|} = \frac{\partial}{\partial\beta}\left(\frac{-1}{\|\mathbf{e}_2^{\text{prev}}\|^2}\frac{\partial}{\partial\alpha}\|\mathbf{e}_2^{\text{prev}}\|\right)$$

$$= \frac{\partial}{\partial\beta}\left(\frac{-1}{\|\mathbf{e}_2^{\text{prev}}\|^2}\right)\cdot\frac{\partial}{\partial\alpha}\|\mathbf{e}_2^{\text{prev}}\| - \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|^2}\frac{\partial^2\|}{\partial\alpha\partial\beta}\mathbf{e}_2^{\text{prev}}\|$$

$$= \frac{2}{\|\mathbf{e}_2^{\text{prev}}\|^3}\frac{\partial}{\partial\beta}\|\mathbf{e}_2^{\text{prev}}\|^2\cdot\frac{\partial}{\partial\alpha}\|\mathbf{e}_2^{\text{prev}}\| - \frac{1}{\|\mathbf{e}_2^{\text{prev}}\|^2}\frac{\partial^2\|}{\partial\alpha\partial\beta}\mathbf{e}_2^{\text{prev}}\|,$$

$$\frac{\partial^2}{\partial\alpha\partial\beta}\|\mathbf{e}_2^{\text{prev}}\| = \frac{\partial}{\partial\beta}\left(\tilde{\mathbf{e}}_{2,0}\cdot\frac{\partial}{\partial\alpha}\mathbf{e}_2^{\text{prev}}\right) = \frac{\partial}{\partial\beta}\tilde{\mathbf{e}}_{2,0}\cdot\frac{\partial}{\partial\alpha}\mathbf{e}_2^{\text{prev}} + \tilde{\mathbf{e}}_{2,0}\cdot\frac{\partial^2\mathbf{e}_2^{\text{prev}}}{\partial\alpha\partial\beta},$$

$$\frac{\partial^2\mathbf{e}_2^{\text{prev}}}{\partial\alpha\partial\beta} = \frac{\partial}{\partial\beta}\left(\mathbb{I}_\alpha - \tilde{\mathbf{e}}_1^\alpha\cdot\tilde{\mathbf{e}}_1\right) = 0.$$

A.3.3 Second order derivatives of η_2

The second derivatives of η_2 are

$$\begin{aligned}\frac{\partial^2\eta_2}{\partial\alpha\partial\beta} &= \frac{\partial^2(\mathbf{S},\mathbf{S})}{\partial\alpha\partial\beta}\cdot\frac{1}{2h(\sigma)}+\frac{\partial(\mathbf{S},\mathbf{S})}{\partial\alpha}\cdot\frac{\partial}{\partial\beta}\frac{1}{2h(\sigma)}\\ &\quad+\frac{\partial(\mathbf{S},\mathbf{S})}{\partial\beta}\cdot\frac{\partial}{\partial\alpha}\frac{1}{2h(\sigma)}+(\mathbf{S},\mathbf{S})\cdot\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{2h(\sigma)},\end{aligned}$$

where we just need to compute the second derivative expressions, since the first derivatives have already been computed in Appendix A.1.4. Hence, ,

$$\begin{aligned}\frac{\partial^2(\mathbf{S},\mathbf{S})}{\partial\alpha\partial\beta} &= \frac{\partial}{\partial\beta}\left(\frac{\partial(\mathbf{S},\mathbf{S})}{\partial\alpha}\right)=\frac{\partial}{\partial\beta}\left(2(\frac{\partial\mathbf{S}}{\partial\alpha},\mathbf{S})\right)=2(\overbrace{\frac{\partial^2\mathbf{S}}{\partial\alpha\partial\beta}}^{0},\mathbf{S})+2(\frac{\partial\mathbf{S}}{\partial\alpha},\frac{\partial\mathbf{S}}{\partial\beta})\\
&= 2(\frac{\partial\mathbf{S}}{\partial\alpha},\frac{\partial\mathbf{S}}{\partial\beta}),\\
\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{2h(\sigma)} &= \frac{\partial}{\partial\beta}\left(-\frac{1}{2h^2(\sigma(\mathbf{S}))}\frac{\partial h(\sigma(\mathbf{S}))}{\partial\alpha}\right)\\
&= \frac{\partial}{\partial\beta}\left(-\frac{1}{2h^2(\sigma(\mathbf{S}))}\right)\cdot\frac{\partial h(\sigma(\mathbf{S}))}{\partial\alpha}-\frac{1}{2h^2(\sigma(\mathbf{S}))}\frac{\partial^2 h(\sigma(\mathbf{S}))}{\partial\alpha\partial\beta}\\
&= \frac{1}{h^3(\sigma(\mathbf{S}))}\frac{\partial h(\sigma(\mathbf{S}))}{\partial\beta}\cdot\frac{\partial h(\sigma(\mathbf{S}))}{\partial\alpha}-\frac{1}{2h^2(\sigma(\mathbf{S}))}\frac{\partial^2 h(\sigma(\mathbf{S}))}{\partial\alpha\partial\beta},\\
\frac{\partial^2 h(\sigma(\mathbf{S}))}{\partial\alpha\partial\beta} &= \frac{\partial}{\partial\beta}\left(\frac{1}{2}\left(1+\frac{\sigma(\mathbf{S})}{\sqrt{\sigma^2(\mathbf{S})+4\delta^2}}\right)\frac{\partial\sigma(\mathbf{S})}{\partial\alpha}\right)\\
&= \frac{1}{2}\left(\frac{1}{\sqrt{\sigma^2(\mathbf{S})+4\delta^2}}-\frac{\sigma(\mathbf{S})}{2}\frac{2\sigma(\mathbf{S})}{(\sigma^2(\mathbf{S})+4\delta^2)^{3/2}}\right)\frac{\partial\sigma(\mathbf{S})}{\partial\beta}\frac{\partial\sigma(\mathbf{S})}{\partial\alpha}\\
&\quad+\frac{1}{2}\left(1+\frac{\sigma(\mathbf{S})}{\sqrt{\sigma^2(\mathbf{S})+4\delta^2}}\right)\frac{\partial^2\sigma(\mathbf{S})}{\partial\alpha\partial\beta}\\
&= \frac{2\delta^2}{(\sigma^2(\mathbf{S})+4\delta^2)^{3/2}}\frac{\partial\sigma(\mathbf{S})}{\partial\beta}\frac{\partial\sigma(\mathbf{S})}{\partial\alpha}+\frac{1}{2}\left(1+\frac{\sigma(\mathbf{S})}{\sqrt{\sigma^2(\mathbf{S})+4\delta^2}}\right)\overbrace{\frac{\partial^2\sigma(\mathbf{S})}{\partial\alpha\partial\beta}}^{0}\\
&= \frac{2\delta^2}{(\sigma^2(\mathbf{S})+4\delta^2)^{3/2}}\frac{\partial\sigma(\mathbf{S})}{\partial\beta}\frac{\partial\sigma(\mathbf{S})}{\partial\alpha},\\
\frac{\partial^2\sigma(\mathbf{S})}{\partial\alpha\partial\beta} &= \frac{\partial^2\det(\mathbf{S})}{\partial\alpha\partial\beta}=\det(\mathbf{W}^{-1})\frac{\partial^2\det(\mathbf{A})}{\partial\alpha\partial\beta}\\
&= \det(\mathbf{W}^{-1})\frac{\partial^2}{\partial\alpha\partial\beta}\left((x_1-x)(y_2-y)-(x_2-x)(y_1-y)\right)\overset{\forall\alpha,\beta=x,y}{=}0.\end{aligned}$$

A.4 Computation of the second order derivatives of the objective function for a submesh

In order to compute the second order derivative of Equation (A.8), we explicit for completeness the first order derivative computed on Appendix A.2,

$$\frac{\partial K_{\hat{\eta_3}}}{\partial \alpha} = \left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^p\right)^{\frac{1}{p}-1} \cdot \sum_{k=1}^{m} \left((\hat{\eta_3}_k)^{p-1} \cdot \frac{\partial \hat{\eta_3}_k}{\partial \alpha}\right).$$

Hence, computing the derivative respect the symbolic variable β, we obtain the second order derivatives:

$$\begin{aligned}\frac{\partial^2 K_{\hat{\eta_3}}}{\partial \alpha \partial \beta} &= \left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^p\right)^{-\frac{p-1}{p}} \sum_{k=1}^{m} \left((\hat{\eta_3}_k)^{p-2} \left(\frac{\partial \hat{\eta_3}_k}{\partial \alpha}\frac{\partial \hat{\eta_3}_k}{\partial \beta} + \frac{\partial^2 \hat{\eta_3}_k}{\partial \alpha \partial \beta}\hat{\eta_3}_k\right)\right) \\ &\quad -(p-1)\left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^p\right)^{\frac{-2p+1}{p}} \left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^{p-1} \frac{\partial \hat{\eta_3}}{\partial \alpha}\right)\left(\sum_{k=1}^{m} (\hat{\eta_3}_k)^{p-1} \frac{\partial \hat{\eta_3}}{\partial \beta}\right).\end{aligned}$$

Bibliography

[Escobar 03] Escobar, J. M., E. Rodríguez, R. Montenegro, G. Montero, and J. M. González-Yuste, *Simultaneous untangling and smoothing of tetrahedral meshes*, Computer Methods in Applied Mechanics and Engineering 192 (25), 2775-2787 (2003).

[Escobar 06] Escobar, J. M., G. Montero, R. Montenegro, and E. Rodríguez, *An algebraic method for smoothing surface triangulations on a local parametric space*, Int. J. Numer. Meth. Engng 66:740-760 (2006).

[Field] Field, David A., *Quality measures for initial meshes*, Int. J. Numer. Meth. Engng. 47, 887-906 (2000).

[Giuliani] Giuliani, S., *An algorithm for continuous rezoning of the hydrodynamic grid in arbitrary Lagrangian-Eulerian computer codes*, Nuclear Engineering and Design, Volume 72, Num. 2 (1982).

[Golberg] M. A. Golberg, *The Derivative of a Determinant*, Mathematical Association of America, The American Mathematical Monthly, Vol. 79, No. 10, pp. 1124-1126 (Dec., 1972).

[Herrman] L. R. Herrmann, *Laplacian-Isoparametric Grid generation scheme*, J. Engng. Mech. Div. 102, 749-756 (1976).

[Knupp 01] Knupp, Patrik M., *Algebraic mesh quality metrics*, SIAM J. Sci. Comput., Vol.23 N1, pp. 193-218 (2001).

[Knupp 03] Knupp, Patrik M., *Algebraic mesh quality metrics for unstructured initial meshes*, Finite Elements in Analysis and Design 39, 217-241 (2003).

[Knupp 03b] Knupp, Patrik M., *A method for hexaedral mesh shape optimization*, Int. J. Numer. Meth. Engng. 58:3119-332 (2003).

[Nocedal] Jorge Nocedal, Stephen J. Wright, *Numerical optimization*, Springer (2006).