Frédéric Alauzet

Metric-Based Anisotropic Mesh Adaptation

Course material, CEA-EDF-INRIA Schools
Numerical Analysis Summer School

June 14-25, 2010
Contents

1 Introduction ................................................................. 1
  1.1 An introducing example ........................................... 1
  1.2 A short history ..................................................... 4
  1.3 Course overview .................................................. 6

2 Metric notion for mesh adaptation ................................. 7
  2.1 Euclidean metric space ........................................... 7
      2.1.1 Geometric interpretation .................................. 9
      2.1.2 Natural metric mapping .................................. 9
  2.2 Riemannian metric space ........................................ 10

3 Metric-based meshing .................................................. 15
  3.1 Generation of adapted anisotropic meshes .................... 15
      3.1.1 Unit element ............................................. 15
      3.1.2 Unit mesh ............................................... 17
      3.1.3 Generating an adapted mesh ............................ 20
  3.2 Operations on metrics ............................................ 21
      3.2.1 Metric Intersection ..................................... 21
      3.2.2 Metric Interpolation ................................... 22
  3.3 Numerical computation of geometric quantities in
      Riemannian metric spaces ..................................... 24
      3.3.1 Computation of lengths .................................. 24
      3.3.2 Computation of volumes ................................. 26
  3.4 Quantifying mesh anisotropy ................................... 26

4 Geometric error estimate ............................................. 29
  4.1 The linear interpolate ........................................... 29
  4.2 A local $L^\infty$ upper bound of the interpolation error ...... 30
  4.3 Metric construction for mesh adaptation ..................... 33
  4.4 Mesh adaptation for analytical functions ................. 36
5 Mesh adaptation for numerical simulations ................. 43
  5.1 Definitions and notations .................................. 43
  5.2 Mesh adaptation scheme .................................... 45
  5.3 Solving the problematic when the solution is unknown ...... 46
  5.4 Numerical computation of the Hessian matrix ............... 47
    5.4.1 Numerical computation of nodal gradients .......... 47
    5.4.2 A double $L^2$-projection method .................. 49
    5.4.3 A method based on the Green formula ............... 49
    5.4.4 A least-square approach ......................... 49
  5.5 Solution Interpolation .................................... 50
    5.5.1 Localization algorithm ............................ 50
      Localization coupled with a grid structure .......... 52
      Localization using the topology of both meshes ...... 53
    5.5.2 Classical polynomial interpolation ................. 54
      Linear interpolation .................................. 54
      Quadratic interpolation .............................. 55
  5.6 Example of a CFD numerical simulation .................... 55

References ....................................................... 59
1

Introduction

Despite the ever-increasing power of computers, reducing the complexity of numerical simulations remains a crucial issue. Mesh adaptation is one among the various methods available to reduce the simulation complexity. The aim is to control the numerical solution accuracy by modifying the discretization of the domain according to size and directional constraints.

In the field of Computational Fluid Dynamics (CFD), unstructured mesh adaptation is well known to reduce significantly the mesh size, i.e., the number of degrees of freedom of the simulation at hand, to reach a given solution accuracy. Thus, it enables substantial gains in CPU time, memory requirement and storage space. The visualization of the results is also facilitated. Furthermore, error estimates have the capability to detect physical phenomena and capture their behavior. Meshes are thus automatically adapted in critical regions without any a priori knowledge of the problem.

This course is an introduction to metric-based anisotropic mesh adaptation.

1.1 An introducing example

This course is motivated by means of a pedagogic example which introduces the basics of anisotropic mesh adaptation. We illustrate the impact of the mesh on the representation of a given function. In the meanwhile, we introduce the notion of anisotropy and of linear interpolate. From this simple study, two questions will arise that will constitute the guideline of this course on mesh adaptation.

Let \( \Omega = [-1, 1] \times [-1, 1] \) be a squared domain of \( \mathbb{R}^2 \) and \( \mathcal{H}_1 \) be a uniform mesh of \( \Omega \) containing 144 vertices. \( \mathcal{H}_1 \) is depicted in Figure 1.1 (left). We consider the function:

\[
\begin{align*}
  f : \quad \mathbb{R}^2 &\rightarrow \mathbb{R} \\
  (x, y) &\mapsto f(x, y) = \sqrt{1-x^2},
\end{align*}
\]
representing a half-cylinder, see Figure 1.1 (middle). We introduce the linear interpolate of \( f \) on \( \mathcal{H}_1 \), denoted \( \Pi_{\mathcal{H}_1} f \), which is the piecewise linear representation of \( f \) on \( \mathcal{H}_1 \), see Figure 1.3 (left). In other words, \( \Pi_{\mathcal{H}_1} f \) is the function that is equal to \( f \) at each vertex of \( \mathcal{H}_1 \) and that is linear inside each triangle. \( \Pi_{\mathcal{H}_1} f \) is represented in Figure 1.1 (right).

Now, if one computes the interpolation error \( e(f) \) of \( f \) on \( \mathcal{H}_1 \), i.e., the gap between \( f \) and its linear interpolate on \( \mathcal{H}_1 \), this error reads:

\[
e(f) = \| f - \Pi_{\mathcal{H}_1} f \|_{L^p(\Omega)}.
\]

For this example, the interpolation error in several norms is given in Table 1.1.

In this context, the problematic of mesh adaptation can be formulated as follow:

_How to reduce the interpolation error with the same number of vertices?_

This can be mathematically written as:

\[
\begin{align*}
\text{Find } & \mathcal{H} = \text{Argmin } \| f - \Pi_{\mathcal{H}} f \|_{L^p(\Omega)} \\
|\mathcal{H}| & = 144
\end{align*}
\]

If the properties of function \( f \) are analyzed, we notice that:

1. the largest variation of the function (curvature) is in direction \( x \)
2. there is no variation of the function in direction \( y \), i.e., the function does not depend on \( y \).

We say that this function is **anisotropic** as it behaves differently in the \( x \) and \( y \) directions. In comparison, the function \( g(x,y) = \exp(-30(x^2 + y^2)) \) which represents a gaussian is isotropic because it behaves (varies) similarly in all directions, see Figure 1.2.
To improve the function representation, we propose to define a new mesh using the anisotropic properties of $f$. Indeed, it is intuitively clear that adding vertices in the $y$-direction is useless and does not improve the solution representation while refining the $x$-axis will provide a more accurate representation. This intuition is illustrated in Figure 1.3 (right). Consequently, we propose to build the mesh having 144 vertices which is the most accurate in the $x$-direction and the less accurate in the $y$-direction meaning that we put uniformly all the vertices on lines $x = -1$ and $x = 1$. This mesh is depicted in Figure 1.4 (left). Visually, $\Pi_{H_2} f$ clearly seems more accurate than $\Pi_{H_1} f$. This fact is confirmed by analyzing the interpolation error in several norms, see Table 1.1. We notice that the interpolation error has been divided by 10 in $L^2$ and $L^\infty$ norms, and by almost 4 in $L^1$ norm!

In this simple example, we have illustrated that generating manually a non-uniform mesh using the anisotropy property of function $f$ allows its representation to be significantly improved. We have manually adapted the mesh.
1. Introduction

Anisotropic mesh adaptation consists in setting up this process automatically. However, we must first answer the following questions:

1. How to communicate with an adaptive mesh generator?
2. How to measure or quantify mesh size and anisotropy from a given function?

In this course, we will show that the fundamental concept of metric is an elegant and very efficient way to answer these questions, hence metric-based mesh adaptation.

<table>
<thead>
<tr>
<th>Norm</th>
<th>$L^1$</th>
<th>$L^2$</th>
<th>$L^\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$|f - \Pi_{H_1}f|_{L^p(\Omega)}$</td>
<td>0.029</td>
<td>0.059</td>
<td>0.133</td>
</tr>
<tr>
<td>$|f - \Pi_{H_2}f|_{L^p(\Omega)}$</td>
<td>0.008</td>
<td>0.005</td>
<td>0.014</td>
</tr>
</tbody>
</table>

Table 1.1. Interpolation error of function $f$ on uniform mesh $H_1$ (top) and adapted mesh $H_2$ (bottom) for several norms.

![Interpolation error of function $f$ on uniform mesh $H_1$ (top) and adapted mesh $H_2$ (bottom) for several norms.](image)

Fig. 1.4. Left, ”adapted” mesh $H_2$. Middle, iso-values of function $f$ on $H_2$. Right, representation of $\Pi_{H_2}f$.

1.2 A short history

The idea of adapting a mesh is very old. A rather large number of papers have been published dealing with mesh adaptation for numerical simulations, e.g. the query ”Mesh Adaptation” on Google Scholar found 395,000 results! In most of these works, the adaptation is isotropic and done by splitting elements thanks to predefined patterns. For instance, a square is split into four squares.
or a triangle is split into four triangles. The most powerful idea of anisotropic mesh adaptation has emerged later at the end of the 80’s due to error estimate and mesh generation concerns.

In 1987, Peraire et al. proposed a first attempt in 2D by providing error measures involving directions [53]. They pointed out the directional property of the interpolation error and the idea to generate elements with aspect ratios. They considered a local mapping procedure to generate elongated elements. They coupled this with an advancing front technique to generate slightly anisotropic meshes, i.e., a 1.5 ratio.

Similar approaches have been considered by Löhner [43], Selmin and Formaggia [56]. The first attempts in 3D were proposed in the early 90’s by Löhner [44] and Peraire et al. [52], but numerical results were almost isotropic and the mesh anisotropy was not clearly visible.

In 1994, Zienkiewicz and Wu [61] gave a qualified status on the subject. Despite some great success with this new approach, they emphasized that: 
"Unfortunately the amount of elongation which can be used in a typical mesh generation by such mapping is small...”.

Almost at the same time on the meshing side, Mavriplis proposed to generate stretched elements with a Delaunay approach in 2D in order to obtain high-aspect ratio triangles in boundary layers and wake regions required by aeronautic numerical simulations [49]. The Delaunay triangulation is performed in a locally stretched space, the idea of metric has almost emerged.

The year after, George, Hecht and Vallet [34] introduced the use of metrics in a Delaunay mesh generator. They exhibited that the absolute value of the hessian is a metric and they proposed a Delaunay-based mesh generator where edge lengths are computed in a Riemannian metric space. This work generalized all the previous ideas.

The fruitful idea of metric was widely exploited for 2D anisotropic mesh adaptation in the 90’s and even more today. For instance, among many others, see the works of [28, 15, 38, 24, 14]. In 1997, Baker gave a state-of-art [8] and wrote: 
"Mesh generation in three dimensions is difficult enough task in the absence of mesh adaptation and it is only recently that satisfactory three-dimensional mesh generators have become available. [...] Mesh alteration in three dimensions is therefore a rather perilous procedure that should be under taken with care”. Indeed, 3D meshing is much more complicated as new pathologies occur and the existence is not guaranteed. Doing 3D anisotropic mesh adaptation is even more complicated.

These blocking points have been partly solved by proposing local remeshing approach to adapt the mesh. These approaches start from an existing mesh and modify it to adapt. Therefore, there is no more any existence problem. At the beginning of the 2000’s, first results with ”really” 3D anisotropic mesh adaptation for applications were published [59, 51, 12, 9, 30, 36, 42].
In the meantime, new more accurate anisotropic error estimates were proposed: a posteriori estimates [54, 25], a priori estimates [26, 4, 39] and goal-oriented estimates for functional outputs [60, 40, 47].

Nowadays, this technology becomes mature and impressive results can be obtained. For instance in [3], the authors report a mean anisotropic ratio of 400 and a mean anisotropic quotient of 50 000 for adapted meshes containing more than 50 millions tetrahedra.

Thanks to its generality, mesh adaptation has been applied to various fields of application and also used with various numerical methods. In all cases, it has brought large improvements in terms of accuracy and CPU performances. Just to give some 3D examples, it has been applied successfully to the sonic boom simulation [3], multi-fluid flows [19, 5], blast problems [2], Stefan problems [9], metal forming processes [13], ... As regards numerical schemes, mesh adaptation has been coupled with Finite Volume [3], Finite Element [5], Stabilized Finite Element [13] Discontinuous Finite Element [55] methods, ...

Nowadays, there are a lot of softwares based on the metric concept. Let us cite Bamg [37] and BL2D [41] in 2D, Yams [29] for discrete surface mesh adaptation and Felflo.a [48], Forge3d [21], Fun3d [40], Gamanic3d [33], MadLib [20], MeshAdap [42], Mmg3d [23], Mom3d [59], Tango [12] and [51] in 3D. It is worth mentioning that all these softwares have arisen from different mesh generation methods. The method in [33, 37] is based on a global constrained Delaunay kernel. In [41], the Delaunay method and the frontal approaches are coupled.

1.3 Course overview

Chapter 2 gives an overview of the relevant concepts in differential geometry which are useful in this course. It begins with a review of the metric spaces and introduces the elementary notations used throughout this course material.

Chapter 3 answers the first question: "how to communicate with an adaptive mesh generator?". It shows that Riemannian metric spaces play a central role in the generation of adapted meshes. Then, operators on metric tensors, which are of main interest in mesh adaptation, are presented. Finally, it details how the numerical evaluations of lengths and volumes are performed in Riemannian metric spaces.

Chapter 4 answers the second question: "how to measure or quantify mesh size and anisotropy from a given function?". It presents a geometric error estimate based on the interpolation error. A metric tensor appears naturally from this error estimate. Then, mesh adaptation is applied to represent at best analytical functions.
Metric notion for mesh adaptation

This Chapter gives an overview of the relevant concepts in differential geometry which are useful in later chapters. It begins with a review of the metric spaces and introduces the elementary notations used throughout this course material.

In order to be self-contained, we recall the differential geometry notions that are used in the sequel. It mainly concerns the computation of lengths in different kinds of spaces: the Euclidean space, Euclidean metric spaces and Riemannian metric spaces. A complete review and the mathematical study of these spaces are available in [10, 11, 22].

We use the following notations in the sequel. Bold face symbols, as \( \mathbf{a}, \mathbf{b}, \mathbf{u}, \mathbf{v}, \mathbf{x}, \ldots \), denote vectors or points of \( \mathbb{R}^3 \). Vector coordinates are denoted by \( \mathbf{x} = (x_1, x_2, x_3) \). The natural dot product between two vectors \( \mathbf{u} \) and \( \mathbf{v} \) of \( \mathbb{R}^3 \) is:

\[
\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{3} u_i v_i.
\]

2.1 Euclidean metric space

An Euclidean metric space \((\mathbb{R}^3, \mathcal{M})\) is a finite vector space where the dot product is defined by means of a symmetric definite positive tensor \( \mathcal{M} \):

\[
\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{M}} = \langle \mathbf{u}, \mathcal{M} \mathbf{v} \rangle = ^{t} \mathbf{u} \mathcal{M} \mathbf{v}, \quad \text{for } (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^3 \times \mathbb{R}^3.
\]

The form \( \mathcal{M} \) is usually written as a \( 3 \times 3 \) matrix that is:

(i) (symmetric) \( \forall (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^3 \times \mathbb{R}^3, \quad \langle \mathbf{u}, \mathcal{M} \mathbf{v} \rangle = \langle \mathbf{v}, \mathcal{M} \mathbf{u} \rangle \)

(ii) (positive) \( \forall \mathbf{u} \in \mathbb{R}^3, \quad \langle \mathbf{u}, \mathcal{M} \mathbf{u} \rangle \geq 0 \)

(iii) (definite) \( \langle \mathbf{u}, \mathcal{M} \mathbf{u} \rangle = 0 \implies \mathbf{u} = 0 \).

These properties ensure that \( \mathcal{M} \) defines a dot product. In the following, the matrix \( \mathcal{M} \) is simply called a **metric tensor** or a **metric**.

**Example 2.1.1.** The most familiar example is maybe the one used by the elementary Euclidean geometry: the three-dimensional **Euclidean space** which is defined by the identity matrix:
The dot product defined by $\mathcal{M}$ makes $\mathbb{R}^3$ become a normed vector space $(\mathbb{R}^3, \| \cdot \|_{\mathcal{M}})$ and a metric vector space $(\mathbb{R}^3, d_{\mathcal{M}}(\cdot, \cdot))$ supplied by the following norm and distance definitions:

- $\forall u \in \mathbb{R}^3$, $\| u \|_{\mathcal{M}} = \sqrt{\langle u, \mathcal{M}u \rangle}$
- $\forall (a, b) \in \mathbb{R}^3 \times \mathbb{R}^3$, $d_{\mathcal{M}}(a, b) = \| b - a \|_{\mathcal{M}} = \| ab \|_{\mathcal{M}}$.

In these spaces, the length $\ell_{\mathcal{M}}$ of a segment $ab = [a, b]$ is given by the distance between its extremities:

$$\ell_{\mathcal{M}}(ab) = d_{\mathcal{M}}(a, b) = \| ab \|_{\mathcal{M}}.$$  \hspace{1cm} (2.1)

Note that this property is generally wrong for a general Riemannian metric space defined hereafter. In an Euclidean metric space, volumes and angles are still well defined. These features are of main interest when dealing with meshing. Given a bounded subset $K$ of $\mathbb{R}^3$, the volume of $K$ computed with respect to metric tensor $\mathcal{M}$ is:

$$|K|_{\mathcal{M}} = \sqrt{\det \mathcal{M}} |K|_{I_3},$$  \hspace{1cm} (2.2)

where $|K|_{I_3}$ is the Euclidean volume of $K$. The angle between two non-zero vectors $u$ and $v$ is defined by the unique real-value $\theta_{\mathcal{M}} \in [0, \pi]$ verifying:

$$\cos(\theta_{\mathcal{M}}) = \frac{\langle u, v \rangle_{\mathcal{M}}}{\| u \|_{\mathcal{M}} \| v \|_{\mathcal{M}}}.$$  \hspace{1cm} (2.3)

In three dimensions, dihedral angles\(^1\) of a tetrahedron can be computed in a given Euclidean space from this definition as it is the angle between the faces normal.

We are also able to compute cross product with respect to metric tensor $\mathcal{M}$:

$$a \times_{\mathcal{M}} b = \mathcal{M}^{\frac{1}{2}} a \times \mathcal{M}^{\frac{1}{2}} b$$

We deduce the following expression of the cross product:

- In 2D: $a \times_{\mathcal{M}} b = \sqrt{\det \mathcal{M}} (a \times b)$
- In 3D: $a \times_{\mathcal{M}} b = \sqrt{\det \mathcal{M}} \mathcal{M}^{-\frac{1}{2}} (a \times b)$

This is very useful in 3D for computing the area of faces with respect to $\mathcal{M}$, let $F$ be a triangle in $\mathbb{R}^3$:

$$|F|_{\mathcal{M}} = \| \mathcal{M}^{\frac{1}{2}} a \times \mathcal{M}^{\frac{1}{2}} b \| = \| a \|_{\mathcal{M}} \| b \|_{\mathcal{M}} \sin(\theta_{\mathcal{M}})$$

or

$$|F|_{\mathcal{M}}^2 = \| \mathcal{M}^{\frac{1}{2}} a \times \mathcal{M}^{\frac{1}{2}} b \|^2 = \| a \|_{\mathcal{M}}^2 \| b \|_{\mathcal{M}}^2 - \langle a, \mathcal{M}b \rangle^2$$

\(^1\) The dihedral angle is the angle between two triangular faces of a tetrahedron.
Finally, as metric tensor $\mathcal{M}$ is symmetric, it is diagonalizable. It thus admits the following spectral decomposition:

$$\mathcal{M} = \mathcal{R} \Lambda^t \mathcal{R},$$

where $\mathcal{R}$ is an orthonormal matrix composed of the eigenvectors $(v_i)_{i=1,3}$ of $\mathcal{M}$:

$$\mathcal{R} = (v_1 \ v_2 \ v_3),$$

verifying $^t\mathcal{R}\mathcal{R} = \mathcal{R}^t\mathcal{R} = \mathcal{I}_3$. $\Lambda = diag(\lambda_i)$ is a diagonal matrix composed of the eigenvalues of $\mathcal{M}$, denoted $(\lambda_i)_{i=1,3}$, which are strictly positive.

### 2.1.1 Geometric interpretation

We will often refer to the geometric interpretation of a metric tensor. In the vicinity $\mathcal{V}(a)$ of point $a$, the set of points that are at distance $\varepsilon$ of $a$, is given by:

$$\Phi_{\mathcal{M}}(\varepsilon) = \{ x \in \mathcal{V}(a) \mid ^t(x - a) \mathcal{M} (x - a) = \varepsilon^2 \}.$$

Note that it is sufficient to describe $\Phi_{\mathcal{M}}(1)$ as $\Phi_{\mathcal{M}}(\varepsilon)$ can be deduced from $\Phi_{\mathcal{M}}(1)$ for all $\varepsilon$ by homogeneity. Another description of $\Phi_{\mathcal{M}}(1)$ can be given using the spectral decomposition $\mathcal{M} = \mathcal{R} \Lambda^t \mathcal{R}$. In the eigenvectors frame, the initial quadratic form $^t(x - a) \mathcal{M} (x - a)$ becomes $^t(\tilde{x} - \tilde{a}) \Lambda (\tilde{x} - \tilde{a})$. Consequently, rewriting $\Phi_{\mathcal{M}}(1)$ in this basis leads to:

$$\Phi_{\mathcal{M}}(1) = \left\{ \tilde{x} \in \mathcal{V}(\tilde{a}) \mid \sum_{i=1}^{n} \lambda_i (\tilde{x}_i - \tilde{a}_i)^2 = 1 \right\}$$

$$= \left\{ \tilde{x} \in \mathcal{V}(\tilde{a}) \mid \sum_{i=1}^{3} \left( \frac{\tilde{x}_i - \tilde{a}_i}{h_i} \right)^2 = 1 \right\}.$$

The last relation defines an ellipsoid centered at $a$ with its axes aligned with the eigen directions of $\mathcal{M}$. Sizes along these directions are given by $h_i = \lambda_i^{-\frac{1}{2}}$. We denote by $\mathcal{B}_M$ this ellipsoid depicted in Figure 2.2. In the sequel, the set $\Phi_{\mathcal{M}}(1)$ is called the unit ball of $\mathcal{M}$.

### 2.1.2 Natural metric mapping

The last information handled by a metric tensor $\mathcal{M}$ is the definition of an application that maps the unit ball $\mathcal{B}_{\mathcal{I}_3}$ of identity metric $\mathcal{I}_3$ onto the unit ball $\mathcal{B}_\mathcal{M}$ of $\mathcal{M}$. We introduce $\mathcal{M}^{-\frac{1}{2}}$ defined by the spectral decomposition:

$$\mathcal{M}^{-\frac{1}{2}} = \mathcal{R} \Lambda^{-\frac{1}{2}} ^t \mathcal{R} \quad \text{where} \quad \Lambda^{-\frac{1}{2}} = diag(\lambda_i^{-\frac{1}{2}}) = \begin{pmatrix} \lambda_1^{-\frac{1}{2}} & 0 & 0 \\ 0 & \lambda_2^{-\frac{1}{2}} & 0 \\ 0 & 0 & \lambda_3^{-\frac{1}{2}} \end{pmatrix}.$$
Basic framework for mesh adaptation
Continuous Metric
Numerical examples
Geometric representation
Unit ball:
\[ \ell(M) = 1 \] and \[ |K| = \sqrt{2} \]

International Meshing Roundtable, Birmingham, 2006

CONTINUOUS METRIC FOR MESH ADAPTATION

This decomposition exists because \( M \) is definite positive. Then, this mapping is given by the application
\[ M^{-\frac{1}{2}} : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \]
\[ x \mapsto M^{-\frac{1}{2}} x. \]

This mapping provides another description of the ellipsoid \( B_M \):
\[ B_M = \{ M^{-\frac{1}{2}} x | \|x\|_2 = 1 \}. \]

\[ \mathcal{I}_2 \quad \mathcal{M}^{-\frac{1}{2}} \]

Fig. 2.1. Geometric interpretation of the unit ball \( B_M = \phi_M(1) \) of \( M \). \( v_i \) are the eigenvectors of \( M \) and \( \lambda_i = h_i^{-2} \) are the eigenvalues of \( M \).

2.2 Riemannian metric space

In differential geometry, a Riemannian manifold or Riemannian space \((M, \mathcal{M})\) is a smooth manifold \( M \) in which each tangent space is equipped with a dot product defined by a metric tensor \( \mathcal{M} \), a Riemannian metric, in a manner which varies smoothly from point to point. Even if no global definition of the scalar product exists, this allows one to define various
geometric notions on a Riemannian manifold such as angles, lengths of curves, areas (or volumes), curvature, gradients of functions and divergence of vector fields. For instance, the distance between two points $x$ and $y$ is given by the length of the curve which locally joins these points along the shortest path: the geodesic. Indeed, Riemannian manifolds are usually curved.

In the context of mesh adaptation, we do not know any manifold, hence any Riemannian manifold. For our concern, we work with a simpler object called a Riemannian metric space defined by $\mathcal{M} = (\mathcal{M}(x))_{x \in \Omega}$. In that specific case, we only know $\mathcal{M}$ a Riemannian metric and $\Omega \subset \mathbb{R}^n$ a common space of parametrization which is our computational domain. There is still no global notion of scalar product.

This mathematical object can be assimilated to a function that can represents a set of Cartesian surface (or graph surface). A Cartesian surface $\mathcal{S}$ of $\mathbb{R}^{n+1}$ is a surface (a sub-manifold of $\mathbb{R}^{n+1}$) locally defined by

$$\mathcal{S} = (x_1, \ldots, x_n, \sigma(x_1, \ldots, x_n)) \in \mathbb{R}^{n+1}$$

with $\sigma$ a $C^2$ scalar function defined on $\mathbb{R}^n$.

Evaluating geometrical quantities in a Riemannian metric space is equivalent to evaluate these quantities on the underlying Cartesian surfaces.

But, we do not need to know such surfaces. In a brought outline, a Riemannian metric space curves the parametrization space, i.e., the computational domain. Let us give an example to illustrate Riemannian metric spaces.

**Example 2.2.1.** This example provides a practical visualization of a Riemannian metric space. Moreover, it illustrates why a Riemannian structure curves space. Let $\mathcal{S} \subset \mathbb{R}^3$ be a Cartesian surface defined by:

$$\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3 | z = \sigma(x, y) = x^2 - y^2 \text{ and } (x, y) \in \Omega\},$$

where $\Omega = \{(x, y) \in \mathbb{R}^2 | 1 \leq x, y \leq 1\}$. This Cartesian surface is represented in Figure 2.3 (left). We consider a curve $\gamma$ drawn in $\Omega$ and parameterized by $t \in \mathbb{R}$:

$$\gamma : [0, 1] \longrightarrow \Omega \subset \mathbb{R}^2 \quad \text{with} \quad t \mapsto \gamma(t) = (x(t), y(t)).$$

Its image $c$ on Cartesian surface $\mathcal{S}$ is a curve of $\mathbb{R}^3$:

$$c : [0, 1] \longrightarrow \mathcal{S} \subset \mathbb{R}^3 \quad \text{with} \quad t \mapsto c(t) = (\gamma(t), \sigma(\gamma(t))) = (x(t), y(t), \sigma(x(t), y(t))).$$

The length of curve $c$ is then given by:

$$\ell(c) = \int_0^1 \left\| \frac{dc}{dt}(t) \right\| \, dt.$$
From the chain rule, we get:

\[
\left\| \frac{dc}{dt}(t) \right\|^2 = \left\| \frac{d\gamma}{dt}(t) \right\|^2 + \left( \frac{d\sigma_{\gamma(t)}}{dt} \left[ \frac{d\gamma}{dt}(t) \right] \right)^2
\]

For \( S \) and by hiding the dependance on \( t \) it yields:

\[
\|dc\|^2 = dx^2 + dy^2 + dz^2 = dx^2 + dy^2 + (d(x^2 - y^2))^2 = dx^2 + dy^2 + (2xdx - 2ydy)^2 = dx^2 + dy^2 + 4x^2dx^2 - 8xydxdy + 4y^2dy^2 = (dx dy)M(x, y)(dx dy)
\]

where

\[
M(x, y) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 4 \begin{pmatrix} x^2 & -xy \\ -xy & y^2 \end{pmatrix} = I_2 + \mathcal{N}.
\]

This quadratic form, \( \|dc\|^2 \), is the first fundamental form of Cartesian surface \( S \) which is represented by \( M \) under matrix form. It exhibits the Riemannian metric space associated with this surface. This Riemannian metric space is shown in Figure 2.3 (right) where the unit ball of the metric at some points of the domain has been drawn. By construction, computing distances in \( (M(x))_{x \in \Omega} \) is equivalent to compute distances on Cartesian surface \( S \subset \mathbb{R}^3 \).

In other words, let \( a \) and \( b \) be two points of \( \Omega \) and \( \gamma(t) = a + t(ab) \) be the straight parametrization of the path of edge \( ab \) in \( \Omega \). Let \( Sa \) and \( Sb \) be the image of these points on \( S \). Then, \( \ell_M(ab) \) is equal to \( \ell_S(SaSb) \) which is the length of the curved path on \( S \) between \( Sa \) and \( Sb \) by following the image of \( \gamma(t) \) by the mapping. In this example, we have considered the Riemannian metric space inherited from the Cartesian parametrization of \( S \) on \( \Omega \). Thus, this surface represents the curvature induced by the Riemannian structure on \( \Omega \).

However, the reciprocity which consists in exhibiting a manifold, \( i.e. \), the Cartesian hypersurface corresponding to the curved domain, from a given Riemannian metric space is a difficult task. Solving this problem requires to seek a hypersurface from its first and second fundamental form [16]. In our case, we only know the first fundamental form.

Now let us extend the notions of length, volume and angle defined in Section 2.1 for Euclidean metric spaces to Riemannian metric spaces which will be the main operations performed by the mesher in such spaces. Fortunately, these notions can be easily derived in the context of meshing because we are not interested in evaluating these quantities on the Riemannian manifold. Indeed, as regards edge length computation, we do not want to compute the distance between two points which requires to find the shortest path on the curved manifold between these two points and to compute the length of the geodesic, but to compute the length of the path between these two points.
2.2 Riemannian metric space

Fig. 2.3. Left, example of a Cartesian surface embedded in $\mathbb{R}^3$. Right, geometric visualization of a Riemannian metric space $(\mathcal{M}(x))_{x \in [0,1] \times [0,1]}$ associated with this surface. At some points $x$ of the domain, the unit ball of $\mathcal{M}(x)$ is drawn.

Defined by the straight line parameterization, i.e., the segment representing the edge between these two points in the parametrization space which is our computational domain. To take into account the variation of the metric along the edge, the edge length is evaluated with an integral formula.

**Definition 2.2.1.** In Riemannian metric space $\mathcal{M} = (\mathcal{M}(x))_{x \in \Omega}$, the length of edge $ab$ is computed using the straight line parameterization in domain $\Omega$

$$\gamma(t) = a + t \, ab,$$

where $t \in [0,1]$:  

$$\ell_{\mathcal{M}}(ab) = \int_0^1 \| \gamma'(t) \|_{\mathcal{M}} \, dt = \int_0^1 \sqrt{\det \mathcal{M}(a + t \, ab) \, ab} \, dt. \quad (2.4)$$

Figure 2.4 depicts iso-values of segment length from the origin for different Riemannian metric spaces. The plotted function is $f(x) = \ell_{\mathcal{M}}(ox)$ where $o$ is the origin of the plane. The iso-values are isotropic for the Euclidean space. They are anisotropic in the case of an Euclidean metric space defined by $\mathcal{M}$. The two principal directions of $\mathcal{M}$ clearly appear. In the case of a Riemannian metric space $(\mathcal{M}(x))_{x \in \Omega}$, all previous symmetries are lost.

To end this section, let us introduce the notions of volume and angle in Riemannian metric spaces.

**Definition 2.2.2.** Given a bounded subset $K$ of $\Omega$, the volume of $K$ computed with respect to Riemannian metric space $(\mathcal{M}(x))_{x \in \Omega}$ is:

$$|K|_{\mathcal{M}} = \int_K \sqrt{\det \mathcal{M}(x)} \, dx. \quad (2.5)$$

For a Riemannian manifold, the angle between two curves is the angle between their tangent vectors at their point of intersection in the tangent plane. In our particular context, edge and tangent vector coincide, hence, we have:
Fig. 2.4. Iso-values of the function $f(x) = \ell_M(ox)$ where $o$ is the origin, i.e., segment length issued from the origin, for different Riemannian metric spaces. Left, in the canonical Euclidean space $([-1, 1] \times [-1, 1], I_2)$, middle, in an Euclidean metric space $([-1, 1] \times [-1, 1], M)$ with $M$ constant and, right, in a Riemannian metric space $(M(x))_{x \in [-1, 1]^2}$ with a varying metric tensor field.

**Definition 2.2.3.** The angle between two edges $pq$ and $pr$ of $\Omega$ in Riemannian metric space $(M(x))_{x \in \Omega}$ is defined by the unique real-value $\theta \in [0, \pi]$ verifying:

$$\cos(\theta) = \frac{\langle pq, pr \rangle_{M(p)}}{\|pq\|_{M(p)} \|pr\|_{M(p)}}. \quad (2.6)$$

In three dimensions, the computation of a dihedral angle in a Riemannian metric space requires to use an integral formula as this angle varies depending on the position on the edge sharing both faces.

**Definition 2.2.4.** The dihedral angle between two faces $[pqr]$ and $[sqp]$ of $\Omega$ sharing common edge $pq$ in Riemannian metric space $(M(x))_{x \in \Omega}$ is defined by the real-value $\theta \in [0, \pi]$ verifying:

$$\cos(\theta) = \int_0^1 \frac{\langle n_{pqr}, n_{sqp} \rangle_{M(p+t\cdot pq)}}{\|n_{pqr}\|_{M(p+t\cdot pq)} \|n_{sqp}\|_{M(p+t\cdot pq)}} \, dt, \quad (2.7)$$

where $p + t \cdot pq$ with $t \in [0, 1]$ is the straight line parameterization of edge $pq$ in domain $\Omega$ and, $n_{pqr}$ and $n_{sqp}$ are the oriented normals to $[pqr]$ and $[sqp]$, respectively.
Metric-based meshing

This Chapter answers the first question of the introductory example: "how to communicate with an adaptive mesh generator?". It shows that Riemannian metric spaces play a central role in the generation of adapted meshes. Then, operators on metric tensors, which are of main interest in mesh adaptation, are presented. Finally, it details how the numerical evaluations of lengths and volumes are performed in Riemannian metric spaces.

3.1 Generation of adapted anisotropic meshes

To generate anisotropic meshes, one must be able to prescribe at each point of the domain a privileged size and orientation of the elements. This information will be transmitted to the mesher which will try to best fit these demands. The use of Riemannian metric spaces is an elegant and efficient way to achieve this goal. Indeed, it is possible for a mesher to work in a Riemannian metric space as the notions of length, volume and angle are well-posed. The main idea of metric-based mesh adaptation, initially introduced in [34], is to generate a unit mesh in a prescribed Riemannian metric space.

There are actually a lot of softwares based on the metric concept. Let us cite **Bang** [37] and **BL2D** [41] in 2D, **Yams** [29] for discrete surface mesh adaptation and **Feflo.a** [48], **Forge3d** [21], **Fun3d** [40], **Gamanic3d** [33], **MAdLib** [20], **MeshAdap** [42], **Mmg3d** [23], **Mom3d** [59], **Tango** [12] and [51] in 3D. It is worth mentioning that all these softwares have arisen from different mesh generation methods. The method in [33, 37] is based on a global constrained Delaunay kernel. In [41], the Delaunay method and the frontal approaches are coupled. [29, 48, 20, 23] are based on local mesh modifications. And [21] is based on the minimal volume principle.

In this section, we first introduce the definition of unit element and then the notion of unit mesh.

3.1.1 Unit element

**Definition 3.1.1.** An element $K$ is unit with respect to a metric $\mathcal{M}$ if the length of all its edges is unit in the metric $\mathcal{M}$.
For instance, a tetrahedron $K$ defined by its list of edges $(e_i)_{i=1..6}$ is unit with respect to $\mathcal{M}$ if:

$$\forall i = 1, ..., 6, \quad \ell_{\mathcal{M}}(e_i) = 1,$$

and its volume is: $|K|_{\mathcal{M}} = \frac{\sqrt{2}}{12}$. Figure 3.1 gives two examples of unit elements for two different metric tensors.

Let $\mathcal{M}$ be a metric tensor, there exists a non-empty infinite set of unit elements with respect to $\mathcal{M}$, as illustrated in Figure 3.2. Conversely, given an element $K$ such that $|K|_{\mathcal{I}} \neq 0$, there is a unique metric tensor $\mathcal{M}$ for which element $K$ is unit with respect to $\mathcal{M}$.

![Fig. 3.1. 3D examples of a unit element with respect to $I_3$ (left) and to an anisotropic metric tensor $\mathcal{M}$ (right). In each case, the unit ball of the metric is drawn at each vertex of the unit element.](image1)

![Fig. 3.2. Several unit elements with respect to a metric tensor in 2D and 3D.](image2)
3.1.2 Unit mesh

The notion of unit mesh is far more complicated than the notion of unit element as the existence of a mesh composed only of unit regular simplexes with respect to a given Riemannian metric space is not guaranteed. For instance, if the Riemannian metric space is not compatible with the domain size, then it clearly does not exist such discrete mesh. To avoid this problem, let us look at the existence of a discrete mesh only made of unit regular simplexes with respect to a Riemannian metric space in $\mathbb{R}^n$. To further simplify the problem, we first consider the Euclidean space $(\mathcal{I}_n(x))_{x \in \mathbb{R}^n}$.

It is well known that $\mathbb{R}^3$ cannot be filled only with the regular tetrahedron while it is possible to fill $\mathbb{R}^2$ with the equilateral triangle. Consequently, even for the simplest continuous mesh $(\mathcal{I}_3(x))_{x \in \mathbb{R}^3}$, there is no discrete mesh composed only of the unit regular tetrahedron. Therefore, the notion of unit mesh has to be relaxed:

**Definition 3.1.2.** A discrete mesh $\mathcal{H}$ of a domain $\Omega \subset \mathbb{R}^n$ is unit for Riemannian metric space $(\mathcal{M}(x))_{x \in \Omega}$ if all its elements are quasi-unit.

Now, let us give a meaning to quasi-unit in three dimensions. A first way to relax the definition of unity is to take into account technical constraints imposed by mesh generators. To converge (and to avoid cycling) while analyzing edges length, the meshing algorithm considers an admissible edge length interval of the form $[\frac{1}{\alpha}, \alpha]$ with $\alpha > 0$, see [29]. If the symmetry property is required, i.e., $\frac{\alpha}{2} = \frac{1}{\sqrt{2}}$, then we obtain $\alpha = \sqrt{2}$. Therefore, as regards the meshing requirement, a tetrahedron $K$ defined by its list of edges $(e_i)_{i=1...6}$ is said to be quasi-unit if $\forall i \in [1, 6]$, $\ell_{\mathcal{M}}(e_i) \in [\frac{1}{\alpha}, \alpha]$. Nevertheless, we do not know if this definition provides the existence of a unit mesh for the Euclidean space $(\mathcal{I}_3(x))_{x \in \mathbb{R}^3}$. In the following, this question of existence is studied by means of the space filling tetrahedra.

*Non-regular space filling tetrahedra.* The study of space filling tetrahedra is an old geometrical question [50, 57]. In the past, it has been demonstrated that there exist sets of non-regular space filling tetrahedra: the Sommerville tetrahedra [58] and the Goldberg tetrahedra families [35].

The Sommerville tetrahedra are built on particular splittings of the unit cube, see Figure 3.3. We recall these tetrahedra thanks to their vertices coordinates. Actually, only the last vertex distinguishes them. $K$ is denoted $(v_1, v_2, v_3, v_4)$ with $v_1 = (0, 0, 0)$, $v_2 = (\frac{1}{2}, -\frac{1}{2}, \frac{1}{2})$, $v_3 = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ and

- $v_4 = (\frac{1}{2}, 0, 0)$ for the Sommerville tetrahedron 1
- $v_4 = (1, 0, 0)$ for the Sommerville tetrahedron 2
- $v_4 = (\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$ for the Sommerville tetrahedron 3
- $v_4 = (\frac{1}{2}, 0, -\frac{1}{2})$ for the Sommerville tetrahedron 4.

The Goldberg tetrahedra are based on the splitting of a prism the basis of which is the equilateral triangle, see Figure 3.4. Their coordinates depend...
on an initial choice of two lengths $a$ and $e$. We specify one of the Goldberg tetrahedra for the specific choice $a = \frac{1}{3}$ and $e = 1$:

- $v_1 = (0, 0, 0)$, $v_2 = (0, 0, 1)$, $v_3 = (0, 1, \frac{1}{3})$ and $v_4 = (\sqrt{\frac{3}{2}}, \frac{1}{2}; \frac{2}{3})$ for the Goldberg tetrahedron.

We suggest now to compare these space filling tetrahedra to the unit regular tetrahedron. To this end, all these tetrahedra are scaled such that their volumes are equal to $\sqrt{\frac{2}{12}}$. The resulting edges lengths for each tetrahedron are specified in Table 3.1. We notice that the proposed notion of quasi-unit element is only verified for Sommerville tetrahedra 1 and 2, and the Goldberg tetrahedron. Therefore, there exists space filling tetrahedra that are quasi-units for the metric $I_3$ in the sense proposed above.

Controlling the volume. Unfortunately, the weaker constraint on the edges length can lead to the generation of quasi-unit elements with a null volume. For instance in $(\mathbb{R}^3, I_3)$, the regular tetrahedron with edges length equal to...
is quasi-unit for $I_3$. However, if one of its vertex is projected orthogonally on the opposite face, then a quasi-unit element of null volume is obtained. Indeed, three edges are of length $\sqrt{2}$ and the three other are of length $\sqrt{6} \approx 0.816 \in \left[ \frac{1}{\sqrt{2}}, \sqrt{2} \right]$. In consequence, controlling only the edges length is not sufficient, the volume must also be controlled to relax the notion of unit element. Practically, this is achieved by managing a quality function:

$$Q_M(K) = \frac{36}{3^3 \sum_{i=1}^{6} \ell_M^2(e_i)} \in [0, 1]$$  \hspace{1cm} (3.1)

For the perfect regular tetrahedron, whatever its edges length, the quality function is equal to 1. For a null volume tetrahedron, $Q_M$ is 0. The qualities of the space filling tetrahedra are given in Table 3.1. Notice that $Q_M$ only quantifies the gap to the regular tetrahedron shape.

We deduce the following definition of quasi-unit element, which is also practically used by mesh generators,

**Definition 3.1.3.** A tetrahedron $K$ defined by its list of edges $(e_i)_{i=1..6}$ is said quasi-unit for $M$ if

$$\forall i \in [1, 6], \; \ell_M(e_i) \in \left[ \frac{1}{\sqrt{2}}, \sqrt{2} \right] \quad \text{and} \quad Q_M(K) \in [\alpha, 1] \quad \text{with} \quad \alpha > 0.$$  

In our case, $\alpha = 0.8$ is an acceptable value as it enables the Sommerville tetrahedra 1 and 2, and the Goldberg tetrahedron to be generated.

**Remark 3.1.1.** Instead of considering $Q_M$, the quality function $\frac{1}{Q_M}$ can be considered. As the variation range becomes $[1, \infty]$, the discrimination of bad elements is made easier.

<table>
<thead>
<tr>
<th>Tetrahedron</th>
<th>Coeff.</th>
<th>Edges length</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sommerville 1</td>
<td>$\sqrt{2}$</td>
<td>0.70 1.22 1.22 1.0 1.41</td>
<td>0.800</td>
</tr>
<tr>
<td>Sommerville 2</td>
<td>$2^\frac{3}{2}$</td>
<td>1.12 0.970 0.970 0.970 0.970 1.12</td>
<td>0.954</td>
</tr>
<tr>
<td>Sommerville 3</td>
<td>$2^\frac{3}{2}$</td>
<td>0.970 0.970 0.970 1.12 1.59 1.12</td>
<td>0.763</td>
</tr>
<tr>
<td>Sommerville 4</td>
<td>$12^{\frac{3}{2}}2^{\frac{3}{2}}$</td>
<td>0.691 1.07 1.07 1.12 1.12 1.23</td>
<td>0.886</td>
</tr>
<tr>
<td>Goldberg</td>
<td>$3^{\frac{3}{2}}2^{\frac{3}{2}}$</td>
<td>0.932 0.990 1.12 1.12 0.990 0.990</td>
<td>0.950</td>
</tr>
</tbody>
</table>

**Table 3.1.** Space filling tetrahedra characteristics. Coeff. is the coefficient that scales the tetrahedron onto a unit volume tetrahedron, i.e., $|K| = \sqrt{2}/12$. The edges length and the tetrahedron quality $Q_Z$ given by Formula (3.1) are provided.
3.1.3 Generating an adapted mesh

To summarize the previous analysis, if \((M(x))_{x \in \Omega}\) is the prescribed Riemannian metric space, then the adaptive mesher (try to) generates a unit mesh of domain \(\Omega\) with respect to \((M(x))_{x \in \Omega}\) according to Definitions 3.1.2 and 3.1.3. The quality of the generated adapted mesh with respect to the perfect unit mesh can be analyzed in terms of size and shape by computing respectively the length of the edges and the quality of the elements in the metric.

Consequently, the generated mesh is uniform and unit in the Riemannian metric space while it is adapted and anisotropic in the Euclidean space. The following pedagogic example gives an intuition of this process.

**Example 3.1.1.** Figure 3.5 shows in background the France map in the meter metric and in foreground the France map in the TGV time metric, i.e., the time spent by the TGV to go from one place to another. The left picture shows how a uniform cartesian grid on the France map in the meter metric is deformed after being mapped onto the France map in the TGV time metric. Exactly the same idea is used in the generation of adapted anisotropic meshes.

![Figure 3.5](image)

**Fig. 3.5.** Two pictures representing the France map in the TGV time metric, i.e., the time spent by the TGV to go from one place to another, as compared to the France in the meter metric.

Let us give a few words about the size prescription by a given Riemannian metric field. We have seen that the size of the unit ball associated with metric tensor \(M\) in the \(i^{th}\) eigenvector direction is given by \(h_i = \lambda_i^{-\frac{1}{2}}\). We then deduce that the prescribed size in direction \(e\) is

\[
h_M(e) = \frac{\|e\|_2}{\ell_M(e)}.
\]

In the case of an Euclidean metric space, the previous relation simply reads

\[
h_M(e) = \frac{\|e\|_2}{\sqrt{e, Me}}.
\] (3.2)
3.2 Operations on metrics

The main advantage when working with metric spaces is the well-posedness of operations on metric tensors, among which the metric intersection and the metric interpolation. These operations have a straightforward geometric interpretation when considering the ellipsoid associated with a metric.

3.2.1 Metric Intersection

When several metrics are specified at a point of the domain, all these metric tensors must be reduced to a single one due to mesh generation concerns. The metric intersection consists in keeping the most restrictive size constraint in all directions imposed by this set of metrics.

Formally speaking, let \( M_1 \) and \( M_2 \) be two metric tensors given at a point. The metric tensor \( M_1 \cap M_2 \) corresponding to the intersection of \( M_1 \) and \( M_2 \) is the one prescribing the largest possible size under the constraint that the size in each direction is always smaller than the sizes prescribed by \( M_1 \) and \( M_2 \). Let us give a geometric interpretation of this operator. Metric tensors are geometrically represented by an ellipse in 2D and an ellipsoid in 3D. But the intersection between two metrics is not directly the intersection between two ellipsoids as their geometric intersection is not an ellipsoid. Therefore, we seek for the largest ellipsoid representing \( M_1 \cap M_2 \) included in the geometric intersection of the ellipsoids associated with \( M_1 \) and \( M_2 \), cf. Figure 3.6, left. The ellipsoid (metric) verifying this property is obtained by using the simultaneous reduction of two metrics.

**Simultaneous reduction.** The simultaneous reduction enables to find a common basis \( (e_1, e_2, e_3) \) such that \( M_1 \) and \( M_2 \) are congruent to a diagonal matrix in this basis, and then to deduce the intersected metric. To do so, the matrix \( N = M_1^{-1}M_2 \) is introduced. \( N \) is diagonalizable with real-eigenvalues. The normalized eigenvectors of \( N \) denoted by \( e_1, e_2 \) and \( e_3 \) constitute a common diagonalization basis for \( M_1 \) and \( M_2 \). The entries of the diagonal matrices, that are associated with the metrics \( M_1 \) and \( M_2 \) in this basis, are obtained with the Rayleigh formula:

\[
\lambda_i = \langle e_i, M_1 e_i \rangle \quad \text{and} \quad \mu_i = \langle e_i, M_2 e_i \rangle \quad \text{for} \quad i = 1\ldots 3.
\]

Let \( P = (e_1, e_2, e_3) \) be the matrix the columns of which are the eigenvectors \( \{e_i\}_{i=1\ldots 3} \) of \( N \). \( P \) is invertible as \((e_1, e_2, e_3)\) is a basis of \( \mathbb{R}^3 \). We have:

\[
M_1 = tP^{-1}
\begin{pmatrix}
\lambda_1 & 0 & 0 \\
0 & \lambda_2 & 0 \\
0 & 0 & \lambda_3
\end{pmatrix}
P^{-1} \quad \text{and} \quad M_2 = tP^{-1}
\begin{pmatrix}
\mu_1 & 0 & 0 \\
0 & \mu_2 & 0 \\
0 & 0 & \mu_3
\end{pmatrix}
P^{-1}.
\]

\[1\] \( \lambda_i \) and \( \mu_i \) are not the eigenvalues of \( M_1 \) and \( M_2 \). They are spectral values associated with basis \((e_1, e_2, e_3)\).
Computing the metric intersection. The resulting intersected metric \( \mathcal{M}_{1 \cap 2} \) is then analytically given by:

\[
\mathcal{M}_{1 \cap 2} = \mathcal{M}_1 \cap \mathcal{M}_2 = t P^{-1} \left( \begin{array}{ccc}
\max(\lambda_1, \mu_1) & 0 & 0 \\
0 & \max(\lambda_2, \mu_2) & 0 \\
0 & 0 & \max(\lambda_3, \mu_3)
\end{array} \right) P^{-1}.
\]

The ellipsoid associated with \( \mathcal{M}_{1 \cap 2} \) is the largest ellipsoid included in the geometric intersection region of the ellipsoids associated with \( \mathcal{M}_1 \) and \( \mathcal{M}_2 \), the proof is given in [1].

Numerically, to compute \( \mathcal{M}_{1 \cap 2} \), the real-eigenvalues of \( \mathcal{N} \) are first evaluated with a Newton algorithm. Then, the eigenvectors of \( \mathcal{N} \), which define \( \mathcal{P} \), are computed using the algebra notions of image and kernel spaces.

Remark 3.2.1. The intersection operation is not commutative. Consequently, when more than two metrics are intersected, the result depends on the order of intersection. In this case, the resulting intersected metric is not anymore optimal. If, we seek for the largest ellipsoid included in the geometric intersection region of several (\( > 2 \)) metrics, the John ellipsoid has to be found thanks to an optimization problem [46].

3.2.2 Metric Interpolation

In practice, the metric field is only known discretely at mesh vertices. The definition of an interpolation procedure on metrics is therefore mandatory to be able to compute the metric at any point of the domain. For instance, the computation of the volume of an element using quadrature formula with Relation (2.5) requires the computation of some interpolated metrics inside the considered element.

Fig. 3.6. Left, view illustrating the metric intersection procedure with the simultaneous reduction in three dimensions. In red, the resulting metric of the intersection of the blue and green metrics. Right, metric interpolation along a segment where the endpoints metrics are the blue and violet ones.
Several interpolation schemes have been proposed in [1] which are based on the simultaneous reduction. The main drawback of these approaches is that the interpolation operation is not commutative. Hence, the result depends on the order in which the operations are performed when more than two metrics are involved. Moreover, such interpolation schemes do not satisfy useful properties such as the maximum principle. Consequently, to design an interpolation scheme on these objects, one needs a consistent operational framework. We suggest to consider the log-Euclidean framework introduced in [7].

**Log-Euclidean framework.** We first define the notion of metric logarithm and matrix exponential. The **metric logarithm** is defined on the set of metric tensors. For metric tensor $\mathcal{M} = \mathcal{R} \Lambda^t \mathcal{R}$, it is given by:

$$\ln(\mathcal{M}) := \mathcal{R} \ln(\Lambda)^t \mathcal{R},$$

where $\ln(\Lambda) = \text{diag}(\ln(\lambda_i))$. The **matrix exponential** is defined on the set of symmetric matrices. For any symmetric matrix $\mathcal{S} = \mathcal{Q} \Xi^t \mathcal{Q}$, it is given by:

$$\exp(\mathcal{S}) := \mathcal{Q} \exp(\Xi)^t \mathcal{Q},$$

where $\exp(\Xi) = \text{diag}(\exp(\xi_i))$. We can now define the **logarithmic addition** $\oplus$ and the **logarithmic scalar multiplication** $\odot$:

$$\mathcal{M}_1 \oplus \mathcal{M}_2 := \exp(\ln(\mathcal{M}_1) + \ln(\mathcal{M}_2))$$

$$\alpha \odot \mathcal{M} := \exp(\alpha \ln(\mathcal{M})) = \mathcal{M}^\alpha.$$

The logarithmic addition is commutative and coincides with matrix multiplication whenever the two tensors $\mathcal{M}_1$ and $\mathcal{M}_2$ commute in the matrix sense. The space of metric tensors, supplied with the logarithmic addition $\oplus$ and the logarithmic scalar multiplication $\odot$ is a vector space.

**Remark 3.2.2.** This framework allows more general computations to be carried out on metric tensors, such as statistical studying or the resolution of PDE’s on metric tensors.

**Metric interpolation in the log-Euclidean framework.** We propose to use the linear interpolation operator derived from the log-Euclidean framework. Let $(\mathbf{x}_i)_{i=1..k}$ be a set of vertices and $(\mathcal{M}(\mathbf{x}_i))_{i=1..k}$ their associated metrics. Then, for a point $\mathbf{x}$ of the domain such that:

$$\mathbf{x} = \sum_{i=1}^k \alpha_i \mathbf{x}_i \quad \text{with} \quad \sum_{i=1}^k \alpha_i = 1,$$

the interpolated metric is defined by:

$$\mathcal{M}(\mathbf{x}) = \bigoplus_{i=1}^k \alpha_i \odot \mathcal{M}(\mathbf{x}_i) = \exp\left(\sum_{i=1}^k \alpha_i \ln(\mathcal{M}(\mathbf{x}_i))\right). \quad \text{(3.3)}$$
This interpolation is commutative, but its bottleneck is to perform \( k \) diagonalizations and to request the use of the logarithm and the exponential functions which are CPU consuming. However, this procedure is essential to define continuously the metric map on the entire domain. Moreover, it has been demonstrated in [7] that this interpolation preserves the maximum principle, i.e., for an edge \( pq \) with endpoints metrics \( M(p) \) and \( M(q) \) such that \( \det(M(p)) < \det(M(q)) \) then we have \( \det(M(p)) < \det(M(p + t \cdot pq)) < \det(M(q)) \) for all \( t \in [0, 1] \).

Accordingly, this metric interpolation enables a continuous metric field to be defined throughout the entire discretized domain. When a metric is required at a point, we determine to which element the point belongs. Then, we apply Relation (3.3), where \( \alpha_i \) are the barycentric coordinates of the point with respect to the element. Figure 3.6 illustrates metric interpolation along a segment, for which the initial data are the endpoints metrics.

Remark 3.2.3. The interpolation formulation (3.3) reduces to

\[
M(x) = \prod_{i=1}^{k} M(x_i)^{\alpha_i},
\]

if all the metrics commute. Therefore, an arithmetic mean in the log-Euclidean framework could be interpreted as a geometric mean in the space of metric tensors.

### 3.3 Numerical computation of geometric quantities in Riemannian metric spaces

Let \( M \) be a discrete metric field defined at the vertices of a mesh \( \mathcal{H} \) of a domain \( \Omega_h \). Thanks to the interpolation operation, we have a continuous metric field in the whole domain, i.e., a Riemannian metric space \( (M(x))_{x \in \Omega_h} \). This representation of the metric field depends on \( \mathcal{H} \) as the interpolation law is applied at the element level.

#### 3.3.1 Computation of lengths

Let \( a \) and \( b \) be two points of the domain. The length of segment \( ab \) is given by Integral (2.4). This integral can be numerically approximated using a \( k \)-points Gaussian quadrature with weights \( (\omega_i)_{i=1...k} \) and barycentric coefficients \( (\alpha_i)_{i=1...k} \):

\[
\ell_{M}(ab) = \int_{0}^{1} \sqrt{t \cdot ab M(a + t \cdot ab) \cdot ab} \, dt \approx \sum_{i=1}^{k} \omega_i \sqrt{t \cdot ab M(a + \alpha_i \cdot ab) \cdot ab}
\]
where $\mathcal{M}(\mathbf{a} + \alpha_i \mathbf{b})$ is the metric at the $i^{th}$ Gauss point.

In the context of meshing, we are interested in the evaluation of edge length in the metric. This is a recurrent operation which could be time consuming for the mesh generator. Fortunately, this operation is at the edge level where the metric is known at the endpoints. The problem can thus be simplified by considering a variation law of the metric along the edge and computing analytically the edge length in the metric.

Formally speaking, let $\mathbf{e} = \mathbf{p}_1 \mathbf{p}_2$ be an edge of the mesh of Euclidean length $\|\mathbf{e}\|_2$, and $\mathcal{M}(\mathbf{p}_1)$ and $\mathcal{M}(\mathbf{p}_2)$ be the metrics at the edge extremities $\mathbf{p}_1$ and $\mathbf{p}_2$. We denote by $\ell_i(\mathbf{e}) = \sqrt{\mathbf{e} \mathcal{M}(\mathbf{p}_i) \mathbf{e}}$ the length of the edge in metric $\mathcal{M}(\mathbf{p}_i)$. We assume $\ell_1(\mathbf{e}) > \ell_2(\mathbf{e})$ and we set $a = \frac{\ell_1(\mathbf{e})}{\ell_2(\mathbf{e})}$.

Now, we consider the metric variation law associated with the interpolation operator on metrics given in Section 3.2.2. The restriction of the (multi-dimensional) metric interpolation operator given by Relation (3.3) to an edge $\mathbf{e} = \mathbf{p}_1 \mathbf{p}_2$ leads to

$$
\lambda(t) = \exp \left( (1 - t) \ln(\lambda_1) + t \ln(\lambda_2) \right) = \lambda_1^{1-t} \lambda_2^t,
$$

which formally yields

$$
\frac{1}{h^2(t)} = \exp \left( \ln \left( \frac{1}{h_1^{2(1-t)} h_2^{2t}} \right) \right),
$$

or

$$
h(t) = h_1^{1-t} h_2^t.
$$

Consequently, this multi-dimensional operator is equivalent to a geometric interpolation law on $h$ on the edge. Note that the geometric interpolation on $\lambda$ is equivalent to the geometric interpolation on $h$.

**Proposition 3.3.1. Assuming that metric field follows a geometric variation law, i.e., a geometric interpolation on $h$: $h(t) = h_1^{1-t} h_2^t$, then we have:**

$$
\ell_{\mathcal{M}}(\mathbf{e}) = \ell_1(\mathbf{e}) \frac{a - 1}{a \ln(a)}.
$$

**Proof.** According to Relation (3.2), we have:

$$
\ell_1(\mathbf{e}) = \frac{\|\mathbf{e}\|_2}{h_1(\mathbf{e})}, \quad \ell_2(\mathbf{e}) = \frac{\|\mathbf{e}\|_2}{h_2(\mathbf{e})}, \quad a = \frac{h_2(\mathbf{e})}{h_1(\mathbf{e})},
$$

and

$$
\ell_{\mathcal{M}}(\mathbf{e}) = \int_0^1 \sqrt{\mathbf{e} \mathcal{M}(t) \mathbf{e}} \, dt = \|\mathbf{e}\|_2 \int_0^1 \frac{1}{h_{\mathcal{M}(t)}(\mathbf{e})} \, dt,
$$

where we have denoted in short $h_{\mathcal{M}(t)} = \mathcal{M}(\mathbf{p}_1 + t \mathbf{p}_1 \mathbf{p}_2)$. For the geometric law on $h$: $h_{\mathcal{M}(t)} = h^{1-t}_1 h^t_2$, it yields

$$
\ell_{\mathcal{M}}(\mathbf{e}) = \|\mathbf{e}\|_2 \int_0^1 \frac{1}{h_1^{1-t}(\mathbf{e}) h_2^t(\mathbf{e})} \, dt = \|\mathbf{e}\|_2 \int_0^1 \frac{1}{h_1(\mathbf{e}) a t} \, dt = \ell_1(\mathbf{e}) \int_0^1 a^{-t} \, dt
$$

$$
= \ell_1(\mathbf{e}) \left[ -\frac{a^{-t}}{\ln(a)} \right]_0^1 = \ell_1(\mathbf{e}) \frac{a - 1}{a \ln(a)}.
$$
Similarly, by considering other laws for the metric variation along the edge, we obtain the following approximations of $\ell_M(e)$:

- **linear variation (arithmetic law) on $h$, $h(t) = (1 - t)h_1 + th_2$:**

  $$\ell_M(e) = \ell_1(e) \frac{\ln(a)}{a - 1},$$

- **linear variation on $\lambda$, $h(t) = ((1 - t)h_1^{-2} + th_2^{-2})^{-\frac{1}{2}}$:**

  $$\ell_M(e) = \ell_1(e) \frac{2a^2 + a + 1}{3a(a + 1)},$$

- **linear variation on $h^{-1}$, $h(t) = ((1 - t)h_1^{-1} + th_2^{-1})^{-1}$:**

  $$\ell_M(e) = \ell_1(e) \frac{a + 1}{2a}.$$

### 3.3.2 Computation of volumes

The evaluation of a tetrahedron volume in a Riemannian metric space consists in computing numerically Integral (2.5). If a first order approximation in the log-Euclidean framework is considered, we get:

$$|K|_M = \int_K \sqrt{\text{det}(M)} dx \approx \sqrt{\text{det} \left( \exp \left( \frac{1}{4} \sum_{i=1}^{4} \log (M_i) \right) \right)} |K|_{x_3}.$$

Higher order approximation can be obtained by using Gaussian quadrature. For instance, if one considers a $k$-point Gaussian quadrature with weights $(\omega_j)_{j=1...k}$ and barycentric coefficients $(\beta_1^j, \beta_2^j, \beta_3^j, \beta_4^j)_{j=1...k}$, it yields:

$$|K|_M \approx |K|_{x_3} \sum_{j=1}^{k} \omega_j \sqrt{\text{det} \left( \exp \left( \sum_{i=1}^{4} \beta_i^j \log (M_i) \right) \right)}.$$

Dihedral angles can be computed using the same methodology.

### 3.4 Quantifying mesh anisotropy

In three dimensions, mesh anisotropy can be quantified by two notions: the **anisotropic ratios** and the **anisotropic quotients**. We first recall both notions and the way they are evaluated numerically. Deriving these quantities for an element relies on the fact that there always exists a unique metric tensor for which this element is unit. If $M_K$ denotes the metric tensor associated with element $K$, solving the following linear system provides $M_K$:
3.4 Quantifying mesh anisotropy

\[
(S) \begin{cases} 
\ell_{M_K}^2(e_1) = 1 \\
\vdots \\
\ell_{M_K}^2(e_6) = 1,
\end{cases}
\]

where \((e_i)_{i=1,6}\) is the edges list of \(K\) and \(\ell_{M_K}^2(e_i) = e_i^T M_K e_i\). \((S)\) admits a unique solution as soon as the volume of \(K\) is not null. Once \(M_K\) is computed, the anisotropic ratio and the anisotropic quotient associated with element \(K\) are simply given by

\[
\text{ratio} = \frac{\min_i \lambda_i}{\max_i \lambda_i} = \frac{\max_i h_i}{\min_i h_i}, \quad \text{and} \quad \text{quo} = \frac{\max_i h_i^3}{h_1 h_2 h_3},
\]

where \((\lambda_i)_{i=1,3}\) are the eigenvalues of \(M_K\) and \((h_i)_{i=1,3}\) are the corresponding sizes. The anisotropic ratio stands for the maximum elongation of a tetrahedron by comparing two eigendirections. The anisotropic quotient represents the overall anisotropic ratio of a tetrahedron taking into account all the possible directions. It corresponds to the overall gain in three dimensions of an anisotropic adapted mesh as compared to an isotropic adapted one. The gain is of course even greater when compared to a uniform mesh.
Geometric error estimate

This Chapter answers the second question of the introductory example: "how to measure or quantify mesh size and anisotropy from a given function?". It presents a geometric error estimate based on the interpolation error. A metric tensor appears naturally from this error estimate. Then, mesh adaptation is applied to represent at best analytical functions.

In this chapter, mesh adaptation is used to modify the mesh $\mathcal{H}$ of domain $\Omega$ in order to better represent a given function $u$ on $\mathcal{H}$. To this end, an error estimate is required, i.e., an estimation of the gap between continuous function $u$ and its discrete representation on $\mathcal{H}$. In the following, we consider the interpolation error estimate.

4.1 The linear interpolate

We first introduce some notations. Let $\Omega_h$ be a discretization of domain $\Omega$ of $\mathbb{R}^d$ and $\mathcal{H}$ a mesh of $\Omega_h$ composed of $N$ vertices. The discretized domain $\Omega_h$ is assumed to be identical to $\Omega$ and thus denoted $\Omega$ in the following. We denote by $\{p_i\}_{i=1..N}$ the set of vertices of mesh $\mathcal{H}$. The approximation space $V_h$ is the space of continuous functions which are piecewise linear (their restriction to each element is linear):

$$V_h = \{ \psi \in C^0(\Omega) \mid \psi|_K \text{ is affine} \forall K \in \mathcal{H} \}.$$  

$V_h$ is also called the usual $P_1$ Finite Element space. The dimension of $V_h$ is equal to the number of mesh vertices, here $N$. A natural basis of $V_h$ is the one formed by the basis functions $(\psi_i)_{1 \leq i \leq N}$. $\psi_i$ is the function of $V_h$ such that:

$$\psi_i(p_j) = \delta_{ij} \quad \text{for } p_j \in \mathcal{H} \iff \begin{cases} \psi_i(p_i) = 1 \\ \psi_i(p_j) = 0 \quad \text{if } j \neq i \end{cases}.$$  

The patch of function $\psi_i$ is the set of triangles sharing vertex $p_i$. These basis functions are also called $P_1$ hat functions. In this basis, any function $v_h$ of $V_h$ reads:

$$v_h(x) = \sum_{i=1}^{N} v(p_i) \psi_i(x).$$  

We call shape functions the restriction of the basis functions to an element.
Definition 4.1.1. The linear interpolate of function \( u \) on mesh \( H \), denoted \( \Pi_h u \), is the piecewise representation of \( u \) on \( H \) given by:

\[
\forall x \in \Omega, \quad \Pi_h u(x) = \sum_{i=1}^{n} u(p_i) \psi_i(x),
\]

where \( \psi_i \) is the \( i \)th shape function. By definition, we have:

\[
\forall p_i \in H, \quad \Pi_h u(p_i) = u(p_i).
\]

4.2 A local \( L^\infty \) upper bound of the interpolation error

Theorem 4.2.1. Let \( K \) be a tetrahedron of mesh \( H \) defined by its list of edges \((e_j)_{j=1...6}\). Let \( u \) be a smooth function two times differentiable and \( H_u \) its Hessian matrix. Then, we have the following bound of the interpolation error in \( L^\infty \) norm:

\[
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{9}{32} \max_{x,y,z \in K} \langle yz, |H_u(x)| yz \rangle \\
\leq \frac{9}{32} \max_{x \in K} \max_{j=1...6} \langle e_j, |H_u(x)| e_j \rangle
\]

where \( |H_u| = R |A| R^{-1} \) with \( |A| = \text{diag}(|\lambda_i|) \).

In other words, a bound of the interpolation error in \( L^\infty \) norm is given by the square length of the largest edge of the tetrahedron computed with respect to the metric of the maximal absolute value of the Hessian:

\[
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{9}{32} \max_{x \in K} \max_{j=1...6} l_{H_u(x)}^2 |e_j|.
\]

(4.1)

Fig. 4.1. Illustration of the linear interpolate \( \Pi_h u \) of a function \( u \).

---

4.2 A local $L^\infty$ upper bound of the interpolation error

Proof. To demonstrate our theorem, we follow the analysis suggested by Anglada et al. [6] on surfaces. We consider a tetrahedron $K$. We note $[a, b, c, d]$ its four vertices and $(e_j)_{j=1 \ldots 6}$ its six edges. Let $u$ be a smooth function two times differentiable and $H_u$ its Hessian matrix. Moreover, we assume that the Hessian of $u$ is definite, i.e., $H_u \neq 0$. Otherwise, $u$ is linear and thus there is no interpolation error.

The aim is to bound the linear interpolation error $e = u - \Pi_h u$ on $K$. We consider a Taylor expansion with integral rest of error function $e$ at one of the vertices of $K$, e.g., $a$, with respect to any interior point $x$ in $K$:

$$
(u - \Pi_h u)(a) = (u - \Pi_h u)(x) + \langle xa , \nabla (u - \Pi_h u)(x) \rangle + \int_0^1 (1 - t) \langle ax , H_u(x + txa) ax \rangle \, dt ,
$$

(4.2)

where $\nabla u(x)$ denotes the gradient of variable $u$ at point $x$. Actually, we assume that the maximal error is achieved at point $x$, then:

$$
\nabla (u - \Pi_h u)(x) = 0
$$

and this is equivalent to:

$$
\langle v, \nabla (u - \Pi_h u)(x) \rangle = 0 , \quad \forall v \subset K .
$$

Furthermore, by definition of the linear interpolate we have: $u(a) = \Pi_h u(a)$.

Thus, Relation (4.2) simplifies to:

$$
|e(x)| = |(u - \Pi_h u)(x)| = \left| \int_0^1 (1 - t) \langle ax , H_u(x + txa) ax \rangle \, dt \right| .
$$

Without loss of generality, we assume that $x$ is closer to $a$ than to $b, c$ or $d$. Let $a'$ represents the point corresponding to the intersection of line $ax$ with the face opposite to $a$. It exists a real number $\lambda$ such that $ax = \lambda aa'$. As $a$ is closer to $x$ than any other vertex of $K$, then $\lambda \leq 3/4$. This yields to write:

$$
|e(x)| = \left| \int_0^1 (1 - t) \lambda^2 \langle aa' , H_u(a + txa) aa' \rangle \, dt \right| ,
$$

$$
\leq \frac{9}{16} \max_{y \in aa'} |(aa' , H_u(y) aa')| \left| \int_0^1 (1 - t) \, dt \right| ,
$$

and then:

$$
|e(x)| \leq \frac{9}{32} \max_{y \in K} |(aa' , H_u(y) aa')| .
$$

Finally, we can introduce the $L^\infty$-norm of the interpolation error:

$$
\|u - \Pi_h u\|_{L^\infty(K)} \leq \frac{9}{32} \max_{y \in K} |(aa' , H_u(y) aa')| .
$$
At this point, it can be of practical interest to introduce a metric tensor in the previous relationship. The $3 \times 3$ Hessian matrix being symmetric, it admits the spectral decomposition $H_u = R \Lambda^t R$ where $R$ and $\Lambda = diag(\lambda_i)$ are the eigenvectors and the eigenvalues matrix, respectively. We introduce the absolute value of the Hessian defined by $|H_u| = R |\Lambda|^t R$ with $|\Lambda| = diag(|\lambda|)$. $|H_u|$ is a metric tensor. We obtain the following bound:

$$
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{9}{32} \max_{y \in K} \langle a a', |H_u(y)| a a' \rangle \\
\leq \frac{9}{32} \max_{y \in K} \max_{v \subset K} \langle v, |H_u(y)| v \rangle .
$$

Notice that the previous relationship is not very useful in practice because of the max. However, we can demonstrate that

$$
\forall v \subset K, \quad \| v \|_{|H_u(x)|} \leq \max_{j=1...6} \| e_j \|_{|H_u(x)|} ,
$$

see [1] for the proof. Previous bound can then be reformulated as follows:

$$
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{9}{32} \max_{y \in K} \max_{j=1...6} \langle e_j, |H_u(y)| e_j \rangle . \quad (4.3)
$$

This expression provides a bound in the case where the maximum error is reached inside element $K$. In the case where the maximum error is achieved on a face or on an edge, a similar proof can be done.

If the maximum error is obtained on element face $F = [a, b, c]$ with edges $(e_j)_{j=1...3}$, then we obtain:

$$
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{2}{9} \max_{y \in K} \max_{j=1...3} \langle e_j, |H_u(y)| e_j \rangle .
$$

Similarly, for a maximum value obtained along edge $e_j$, we have:

$$
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{1}{8} \max_{y \in e_j} \langle e_j, |H_u(y)| e_j \rangle .
$$

In conclusion, as $\frac{1}{8} < \frac{2}{9} < \frac{9}{32}$, Relation (4.3) provides a proper bound for the interpolation error on element $K$.

To demonstrate Relation (4.1), we use the definition of the length in a metric space, see Relation (2.1). We have: $\ell^2_{|H_u(x)|}(e_j) = \langle e_j, |H_u(y)| e_j \rangle$. Consequently, the previous bound can be rewritten:

$$
\| u - \Pi_h u \|_{L^\infty(K)} \leq \frac{9}{32} \max_{x \in K} \max_{j=1...6} \ell^2_{|H_u(x)|}(e_j) .
$$

**Remark 4.2.1.** The constant appearing in the interpolation error bound can be evaluated for any dimension by induction. In dimension $d$, we have:

$$
c_d = \lambda^2 \left| \int_0^1 (1 - t) \, dt \right| = \frac{1}{2} \left( \frac{d}{d+1} \right)^2 .
$$
Remark 4.2.2. Theorem 4.2.1 allows us to estimate the interpolation error on any element $K$:

$$
\varepsilon_K = c_d \max_{y \in K} \max_{j=1\ldots 6} \langle e_j, |H_u(y)| e_j \rangle,
$$

where $\varepsilon_K$ is the estimated error on $K$.

4.3 Metric construction for mesh adaptation

Let us give a short synthesis of the previous chapters. We have seen that:

1. metric tensor $M$ can be used to prescribe sizes directionally to the mesher
2. to generate an adapted mesh $\mathcal{H}$, the mesher generates a unit mesh with respect to Riemannian metric space $(M(x))_{x \in \Omega}$
3. the previous theorem provides an estimate of the interpolation error of function $u$ on an element $K$.

Now, we aim at generating an anisotropic adapted mesh to control the interpolation error model. How can we link these three items?

To this end, we fix the error threshold to $\varepsilon$ and we want to commit an error of $\varepsilon$ on each element $K$ of mesh $\mathcal{H}$:

$$
\forall K \in \mathcal{H}, \ c_d \max_{x \in K} \max_{j=1\ldots 6} \langle e_j, |H_u(x)| e_j \rangle = c_d \max_{x \in K} \max_{j=1\ldots 6} \ell^2_{|H_u(x)|}(e_j) = \varepsilon.
$$

However, the left-hand side term remains hard to compute numerically due to the max$_{x \in K}$ on the Hessian. Indeed, the only information we have on the Hessian is a discrete one: the pointwise values at mesh vertices $\{|H(p_i)|\}_{p_i \in \mathcal{H}}$. However, the max$_{x \in K}$ operator requires the knowledge of $H_u(x)$ everywhere inside tetrahedron $K$, i.e., a continuous information. Thanks to the metric interpolation operator, Section 3.2.2, we are able to define Riemannian metric space $(|H(x)|)_{x \in \Omega}$. We thus propose to control the error by generating a mesh $\mathcal{H}$ such that:

$$
\forall K \in \mathcal{H}, \ c_d \max_{x \in K} \ell^2_{|H_u|}(e_j) = \varepsilon.
$$

This relation points out that it is sufficient to control the length of the mesh edges in Riemannian metric space $(|H(x)|)_{x \in \Omega}$ to control the interpolation error on each element of the mesh. Our problem of mesh adaptation reduces to, generate a mesh $\mathcal{H}$ such that:

$$
\forall e \in \mathcal{H}, \ c_d \ell^2_{|H_u|}(e) = \varepsilon.
$$

Notwithstanding, we still have to make the link with the notion of unit mesh. As we have:

$$
c_d (e, |H_u| e) = \varepsilon \iff (e, \frac{c_d |H_u|}{\varepsilon} e) = 1,
$$
the mesh adaptation problem can finally be formulated as: Generate a unit mesh $H$ in Riemannian metric space $(\mathcal{M}(x))_{x \in \Omega} = (\frac{c_d |H_u|}{\varepsilon}(x))_{x \in \Omega}$:

$$\forall e \in H, \quad \ell^2_{\mathcal{M}}(e) = 1.$$ 

This mesh will be optimal to control the interpolation error in $L^\infty$-norm.

The mesh adaptation strategy described above consists in equidistributing the interpolation error over the mesh.

**Remark 4.3.1.** Let us give a concrete geometric interpretation of this result.

If one considers a linear function in 1D (a line), then only two points are needed to exactly represent this function. On the other hand, in the case of a curve, the number of points which are required to describe exactly this curve in not finite in general. To describe this curve with a given accuracy, it is mandatory to know its intrinsic properties, notably its local curvature. The points density must be locally proportional to the curvature. Therefore, we need a control based on the second derivative of the function. In other words, we do not adapt to the function slope (its gradient) but to the function curvature (its Hessian). This error estimate measures the maximal gap between the curve and its piecewise linear representation on the mesh.

In 2D, the function can be interpreted as a Cartesian surface embedded in $\mathbb{R}^3$: $\Sigma = (x, y, u(x,y))$, see Figure 4.2. Analogously with the dimension one, the error estimate measures the gap between the underlying surface $\Sigma$ and its piecewise linear approximation on the mesh defined by $\Pi_h u$. It imposes a point density proportional to the principal curvatures of the surface. In 3D, the function can be perceived as a Cartesian hypersurface embedded in $\mathbb{R}^4$.

This is the justification for the term of geometric error estimate, indicating also that it is problem-independent.

**Remark 4.3.2.** It is important to distinguish the Cartesian (hyper)surface representing the solution and a manifold that has $(\mathcal{M}(x))_{x \in \Omega}$ as natural Riemannian metric (the first fundamental form).

**Handling degenerated cases.**

In practice, the Hessian of function $u$ can be 0, e.g. if $u$ is linear, then $|H_u|$ is not definite. In this particular case, the interpolation error is 0 and we want to prescribe a mesh size which is infinite. To solve this issue, this infinite size prescription is truncated by imposing a maximal size $h_{\text{max}}$. This is equivalent to truncate tiny eigenvalues by $\lambda_{\text{min}} = h_{\text{max}}^{-2}$. The metric is then defined by:

$$\mathcal{M} = \mathcal{R} \tilde{\Lambda}^t \mathcal{R} \quad \text{where} \quad \tilde{\Lambda} = \text{diag}(\tilde{\lambda}_i),$$

and

$$\tilde{\lambda}_i = \max \left( \frac{c_d |\lambda_i|}{\varepsilon}, h_{\text{max}}^{-2} \right),$$
the \((\lambda_i)_i\) being the eigenvalues of \(H_u\).

Similarly, if one wants to limit the accuracy of the adapted mesh, a minimal size \(h_{\text{min}}\) is imposed to truncate to small size prescription. This is equivalent to truncate large eigenvalues by a \(\lambda_{\text{max}} = h_{\text{min}}^{-2}\). The metric is then defined by:

\[
\mathcal{M} = \mathcal{R} \tilde{A}^t \mathcal{R} \quad \text{where} \quad \tilde{A} = \text{diag}(\tilde{\lambda}_i),
\]

and

\[
\tilde{\lambda}_i = \min \left( \max \left( \frac{c_d |\lambda_i|}{\varepsilon}, h_{\text{max}}^{-2} \right), h_{\text{min}}^{-2} \right).
\]

Isotropic mesh adaptation.

Relation (4.4) provides an anisotropic expression of the metric. If an isotropic size prescription is expected, the isotropic metric is obtained by taking into account the smallest size prescription, i.e., the largest eigenvalue. This metric is given by:

\[
\mathcal{M}_{\text{iso}} = \text{diag}(\max_i(\tilde{\lambda}_i)) = \begin{pmatrix}
\max_i(\tilde{\lambda}_i) & 0 & 0 \\
0 & \max_i(\tilde{\lambda}_i) & 0 \\
0 & 0 & \max_i(\tilde{\lambda}_i)
\end{pmatrix}.
\]

Fig. 4.2. Left, a 2D function and, right, its representation as a Cartesian surface in \(\mathbb{R}^3\).
4.4 Mesh adaptation for analytical functions

The mesh adaptation strategy described in the previous sections is applied to represent at best a given function \( f \) on \( \mathcal{H} \) at a given budget, \textit{i.e.}, with a prescribed number of vertices. This mesh adaptation problem is non-linear. We therefore propose the following iterative algorithm to solve it:

**Initial mesh** (\( \mathcal{H}^0 \)) and set error threshold \( \varepsilon \)

//--- Adaptive loop
For \( i=1, \text{nadap} \)

1. \( (f_{i-1}, H_{f_{i-1}}) = \) Evaluate \( f \) and its Hessian \( H_f \) on mesh \( \mathcal{H}_{i-1} \)

2. \( M_{i-1} = \) Compute metric \( M \) according to Relation (4.4)

3. \( \mathcal{H}_i = \) Generate a new adapted mesh from pair \( (\mathcal{H}_{i-1}, M_{i-1}) \)

End for

**The \( \chi \)-shaped function:** We consider a square domain \( \Omega = [0, 1] \times [0, 1] \) and function \( f_1 \) defined by:

\[
f_1(x, y) = \tanh(-100(y - 0.5 - 0.25\sin(2\pi x))) + \tanh(100(y - x)) .
\]

This function is represented in Figure 4.4. The aim is to obtain the best adapted mesh containing 5,000 vertices. We start from a uniform mesh \( \mathcal{H}_{\text{unif}} \) composed of 5,417 vertices and 10,585 triangles, Figure 4.3 (left). We set the error threshold to \( \varepsilon = 0.04 \) and after 6 mesh adaptation iterations, we obtain

![Fig. 4.3. Left, uniform initial mesh and, right, final anisotropic adapted mesh for function \( f_1 \).](image-url)
Fig. 4.4. Representation of function $f_1$ on uniform initial mesh (left) and on final anisotropic adapted mesh (right).
an anisotropic adapted mesh $\mathcal{H}_{\text{adap}}$ composed of 5132 vertices and 10084 triangles, depicted in Figure 4.3 (right).

In Figure 4.4, the representation of function $f_1$ on the adapted mesh (right), i.e., $\Pi_{\mathcal{H}_{\text{adap}}}f_1$, is visually more accurate that its representation on the uniform mesh (left) with the same number of vertices. Mathematically, we can evaluate an approximation of the interpolation error in $L^\infty$-norm on each mesh, we get:

\[
\|f_1 - \Pi_{\mathcal{H}_{\text{unif}}}f_1\|_{L^\infty(\Omega)} \approx 4.63 \times 10^{-1} \\
\|f_1 - \Pi_{\mathcal{H}_{\text{adap}}}f_1\|_{L^\infty(\Omega)} \approx 6.74 \times 10^{-3} .
\]

The interpolation error has been reduced by a factor 70 which confirms our visual intuition.

The 5-wells function: We consider a square domain $\Omega = [0,1] \times [0,1]$ and function $f_2$ defined by:

\[
f_2(u,v) = \tanh\left(30(u^2 + v^2 - \epsilon)\right)
+ \tanh\left(30((u - 0.75)^2 + (v - 0.75)^2 - \epsilon)\right)
+ \tanh\left(30((u + 0.75)^2 + (v - 0.75)^2 - \epsilon)\right)
+ \tanh\left(30((u + 0.75)^2 + (v + 0.75)^2 - \epsilon)\right)
\]

with $\epsilon = 0.25$ and $u = 4x - 2$, $v = 4y - 2$.

This function is represented in Figure 4.6. The aim is again to obtain the best adapted mesh containing 5000 vertices. We start from the same uniform mesh $\mathcal{H}_{\text{unif}}$ composed of 5417 vertices and 10585 triangles, Figure 4.3 (left). We set the error threshold to $\epsilon = 0.15$ and after 6 mesh adaptation iterations, we obtain an anisotropic adapted mesh $\mathcal{H}_{\text{adap}}$ composed of 5029 vertices and 10013 triangles, depicted in Figure 4.5 (right).
Fig. 4.6. Representation of function $f_2$ on uniform initial mesh (left) and on final anisotropic adapted mesh (right).
In Figure 4.4, the representation of function $f_2$ on the adapted mesh (right), i.e., $\Pi_{\mathcal{H}_{\text{adapt}}} f_2$, is visually more accurate that its representation on the uniform mesh (left) with the same number of vertices. Notably, in regions where wells intersect. Mathematically, we can evaluate an approximation of the interpolation error in $L^\infty$-norm on each mesh, we get:

\[
\|f_2 - \Pi_{\mathcal{H}_{\text{unif}}} f_2\|_{L^\infty(\Omega)} \approx 3.40 \times 10^{-1}
\]
\[
\|f_2 - \Pi_{\mathcal{H}_{\text{adapt}}} f_2\|_{L^\infty(\Omega)} \approx 3.43 \times 10^{-2}
\]

The interpolation error has been reduced by a factor 10 which confirms our visual intuition.

Comparison between isotropic and anisotropic mesh adaptation: To finish with these analytical examples, we compare anisotropic mesh adaptation given by Relation (4.4) and isotropic mesh adaptation given by Relation (4.5). Adapted isotropic meshes for both functions contain almost 5000 vertices and are depicted in Figure 4.7. The interpolation errors are:

\[
\begin{align*}
\|f_2 - \Pi_{\mathcal{H}_{\text{unif}}} f_2\|_{L^\infty(\Omega)} &\approx 3.40 \times 10^{-1} \\
\|f_2 - \Pi_{\mathcal{H}_{\text{adapt}}} f_2\|_{L^\infty(\Omega)} &\approx 3.43 \times 10^{-2}
\end{align*}
\]

\textbf{Fig. 4.7.} Final isotropic (left) and anisotropic (right) adapted meshes for functions $f_1$ (top) and $f_2$ (bottom).
\[ \| f_1 - \Pi_{H_{iso}} f_1 \|_{L^\infty(\Omega)} \approx 9.52 \times 10^{-2} \]
\[ \| f_2 - \Pi_{H_{iso}} f_2 \|_{L^\infty(\Omega)} \approx 8.90 \times 10^{-2}. \]

These errors indicate the significant gains in accuracy that are expected by using anisotropic adapted meshes instead of isotropic adapted ones.
Mesh adaptation for numerical simulations

This chapter focuses on the application of mesh adaptation to numerical simulations. It describes how to compute the metric when the solution of the problem is not known. And finally, it explains how to couple mesh adaptation with flow solver.

This chapter focuses on the application of mesh adaptation to numerical simulations. In this case, we face two main problematics:

1. the solution $u$ of the problem is not known, so does its linear interpolate $\Pi_h u$ and its Hessian $H_u$. We only know the numerical solution $u_h$ and we want to control the approximation error $\|u - u_h\|$, see Figure 5.3 (left)
2. the mesh adaptation process has to be coupled with the flow solver.

The first point means that the geometric error estimate presented in Chapter 4 cannot be applied directly. A complementary treatment is added or complementary assumption are made to exhibit the metric from the numerical solution $u_h$.

The second point arises once a new adapted mesh has been generated. Indeed, the numerical solution is not known on this new mesh. One can restart the simulation from scratch but this will significantly degrade the efficiency of the method in term of CPU time. To solve this issue, we introduce a new step: the interpolation stage. It consists in projecting the computed solution on this new mesh. Then, the flow solver restart the simulation from the last state, hence avoiding to waste CPU time.

This chapter first describes the mesh adaptation scheme in the context of steady flow. Then, the two problematics are addressed. A numerical example closes this chapter.

5.1 Definitions and notations

Let us consider a bounded domain $\Omega \in \mathbb{R}^2$ and let $\partial \Omega$ be its boundary. We like to introduce a triangular mesh $\mathcal{H} = \bigcup K_i$ of domain $\Omega$ composed with triangles. A triangle in $\mathbb{R}^2$ is defined by the list of its vertices which are locally numbered in a convenient way. This list, enriched with some conventions, provides the complete definition of the related element, including the definition
of its edges and neighbors, together with an orientation. Indeed, in our applications, we strictly require an orientation of the mesh elements. In particular, the oriented local numbering of the triangle vertices enables us to compute its surface area while giving a sense to its sign. It also enables directional normals to be evaluated for each edge.

Formally speaking, the local numbering of vertices, edges and neighboring triangles is pre-defined in such a way that some properties are implicitly induced. This definition is only a convenient convention resulting in implicit properties. In the case of a triangle with vertices \([p_0, p_1, p_2]\) in this order, the first vertex having been chosen, the numbering of the others is deduced counter-clockwise, see Figure 5.1 (left). This orientation provides us with positive sign while computing the triangle surface area. Then, the topology can now be well defined thanks to the edges definition:

\[
e_0 = p_1p_2, \quad e_1 = p_2p_0 \quad \text{and} \quad e_2 = p_0p_1.
\]

This numeration is such that the index of the edge is the index of the viewing vertex, i.e., the opposite vertex. Regarding the neighboring triangles, we denote by \(K_i\) the neighbor viewing vertex \(p_i\) through edge \(e_i\), see Figure 5.1 (left).

In the following all the indices in square bracket are given modulo 3 : \([i] = i \mod(3)\). Let \(K = [p_0, p_1, p_2]\) be a triangle, its signed (surface) area \(A_K\) is given by:

\[
A_K = \frac{1}{2} \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = \frac{1}{2} \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix},
\]

where \(p_i = (x_i, y_i)\). This area is positive if the triangle is numbered counter-clockwise which is our convention on the mesh orientation. The signed area is also given by one half of the \(z\)-component of \(p_0p_1 \wedge p_0p_2\).

Let \(p\) be a point, we denote by \(K^i\) the virtual triangle where vertex \(p_i\) is substituted by \(p\). The signed areas \(A_{K^i}\), for \(i = 0 \ldots 2\), are called the barycentrics.

---

**Fig. 5.1.** Left, definition of triangle \(K\) and its three neighbors \(K_i\). Vertices indices are ordered counter-clockwise and the neighboring entities numeration is the same as the viewing vertices. Right, the seven regions defined by the signs of the three barycentric coordinates of a point \(p\) with respect to element \(K\).
of \( p \). The three associated **barycentric coordinates** are given by:

\[
\beta_i = \frac{A_{Ki}}{A_K} \quad \text{for} \quad i = 0 \ldots 2.
\]

The sign of the three barycentric coordinates or barycentrics defines explicitly seven regions of the plane where point \( p \) can be located with respect to element \( K \). The possible combinations are given in Figure 5.1 (right).

We denote by \( \{p_i\}_{i=1}^N \) the set of vertices of mesh \( \mathcal{H} \). The numerical solution \( u_h \) lies in the approximation space \( V_h \) of the continuous functions which are piecewise linear:

\[
V_h = \{ \psi \in C^0(\Omega) \mid \psi|_K \text{ is affine } \forall K \in \mathcal{H} \}.
\]

We denote by \( (\psi_i)_{1 \leq i \leq N} \) the basis function of \( V_h \). We recall that, in this basis, \( u_h \in V_h \) writes:

\[
u_h(x) = \sum_{i=1}^{N} v(p_i) \psi_i(x), \quad \forall x \in \Omega_h.
\]

Let \( K \) be a triangle and \([p_0, p_1, p_2]\) its three vertices. If \( x \in K \), then we have:

\[
u_h(x) = u_h(p_0) \psi_0(x) + u_h(p_1) \psi_1(x) + u_h(p_2) \psi_2(x), \quad (5.1)
\]

where \( \psi_i(x) = \beta_i = \frac{A_{Ki}}{A_K} \) is the \( i \)th the barycentric coordinates of \( x \) with respect to \( K \).

### 5.2 Mesh adaptation scheme

Mesh adaptation is a non-linear problem linked to the solution. Therefore, it seems intuitive to propose an iterative procedure to solve this problem. For steady simulations, an adaptive computation is carried out via a mesh adaptation loop inside which an algorithmic (or iterative) convergence of the couple mesh-solution is sought, in the sense that the solution is algorithmically (or iteratively) converging toward the steady state solution and the mesh is converging toward the adapted mesh associated to this converged steady state.

This iterative scheme is illustrated in Figure 5.2, where \( i \) indicates the adaptation iteration index and where \( \mathcal{H}, \mathcal{S}, S^0 \) and \( \mathcal{M} \) denote the mesh, the solution, the initial solution at each iteration and the metric, respectively. At each stage, a numerical solution is computed on the current mesh and is analyzed by means of the error estimate. In our case, the metric is obtained via the geometric error estimate. Next, a unit mesh is generated with respect to this metric. Finally, the solution is interpolated on the new adapted mesh.

This procedure is repeated until the algorithmic convergence of the solution and of the mesh is achieved.
5.3 Solving the problematic when the solution is unknown

As mentioned before, the solution $u$ of the problem is not known, so does its linear interpolate $\Pi_h u$ and its Hessian $H_u$. We only know the numerical solution $u_h$ and we want to control the approximation error $\|u - u_h\|$. For elliptic problems, Céa’s lemma [17] states that the approximation (finite element) error can be bounded by the interpolation error:

$$\|u - u_h\| \leq c \|u - \Pi_h u\|,$$

where $\|\cdot\|$ is a norm of $\mathbb{R}^d$ and $c$ a constant independent of the current mesh.

Here, we assume that this relation still holds in the class of problems envisaged. Actually, similar analysis based on the interpolation error show (practically) that the link between the interpolation error and the approximation error is even stronger than the bound given by Céa’s lemma. The interpolation error is indeed a reasonable way of defining an error estimate according to [27].

Nevertheless, our analysis cannot be applied directly as $u$ is not known. The idea is to build a higher order solution approximation $u^*$ of $u$ from $u_h$ which is twice continuously differentiable and to consider $u^*$ in our error estimate. We do not assume that $u^*$ is a better approximation of $u$ than $u_h$ but $u^*$ will be built in such a way that its derivatives better approximates the derivatives of $u$. In other words, the interpolation error is approximated as:

$$\|u - \Pi_h u\| \approx \|u^* - \Pi_h u^*\|.$$

Practically, only the Hessian of $u^*$ is recovered.
In the context of discontinuous flows, the numerical solution is also piecewise linear by elements even if it approximates a discontinuous solution. The mesh acts as a regularization operator on the solution. In this case, we still approximate the solution $u$ with a continuous higher order representation and we still apply our error estimate.

### 5.4 Numerical computation of the Hessian matrix

One of the key points in the mesh adaptation scheme is related to the construction of the metric tensor and, more precisely, to the evaluation of the Hessian matrix. Indeed, constructing a good (reasonable) metric requires computing the Hessian matrix accurately. Various techniques have been investigated so far. In the following, we present three of them.

#### 5.4.1 Numerical computation of nodal gradients

Let $K$ be a triangle and $[p_0, p_1, p_2]$ its three vertices. If $x \in K$, Relation (5.1) provides the values $u_h(x)$. Similarly, the gradients $\nabla u_h(x)$ is given by:

$$ \nabla u_h(x) = u_h(p_0) \nabla \psi_0(x) + u_h(p_1) \nabla \psi_1(x) + u_h(p_2) \nabla \psi_2(x), \quad (5.2) $$

where

\[
\begin{align*}
\frac{\partial \psi_0}{\partial x} &= \frac{y_1 - y_2}{2A_K} & \frac{\partial \psi_0}{\partial y} &= \frac{x_2 - x_1}{2A_K}, \\
\frac{\partial \psi_1}{\partial x} &= \frac{y_2 - y_0}{2A_K} & \frac{\partial \psi_1}{\partial y} &= \frac{x_0 - x_2}{2A_K}, \\
\frac{\partial \psi_2}{\partial x} &= \frac{y_0 - y_1}{2A_K} & \frac{\partial \psi_2}{\partial y} &= \frac{x_1 - x_0}{2A_K}.
\end{align*}
\]

Fig. 5.3. Left, illustration of exact solution $u$ (black), its linear interpolate $\Pi_h u$ (red) and numerical solution $u_h$ (green). Right, illustration of piecewise constant gradient $\nabla u_h$ (blue) and piecewise linear recovered gradient $\nabla u^*$ (purple).
We deduce that $\nabla u_h$ is piecewise constant, i.e., constant on each element $K$. We denote by $\nabla u_h|_K$ the constant gradient on element $K$.

However, $\nabla u_h$ is not defined at mesh vertices. We propose to recover nodal gradients for the piecewise constant representation of $\nabla u_h$ using a $L^2$-projection method. The local $L^2$-projection operator is based on the Clément interpolation operator [18].

Let $p_i$ be a vertex of mesh $\mathcal{H}$. We denote by $S_i$ the stencil of basis function $\psi_i$, i.e., $S_i = \text{supp } \psi_i$, which is in fact the topological ball of $p_i$. We introduce the approximation spaces of the piecewise constant and linear function:

$$V^0_h = \{ v \in L^2(\Omega) \mid v|_K \in \mathbb{P}^0 \text{ } \forall K \in \mathcal{H} \}$$

$$V^1_h = \{ v \in C^0(\Omega) \mid v|_K \in \mathbb{P}^1 \text{ } \forall K \in \mathcal{H} \}.$$ 

where $\mathbb{P}^0$ and $\mathbb{P}^1$ are the set of constant and linear polynomials. The idea is to find, in a $L^2$-norm sense, the best constant gradient $\nabla R u_h$ approximating the piecewise constant field $\nabla u_h$ on $S_i$. More precisely, for $v \in L^2(\Omega)$, we define $\Pi_0 v \in V^0_h$ by:

$$\forall S_i \subset \mathcal{H}, \begin{cases} (\Pi_0 v)|_{S_i} \in \mathbb{P}^0 \\ \int_{S_i} (\Pi_0 v - v) w = 0 & \forall w \in \mathbb{P}^0. \end{cases}$$

We then define the Clément interpolation operator $\Pi_c : V^0_h \rightarrow V^1_h$:

$$\Pi_c v := \sum_{i=1}^{n} \Pi_0 v(p_i) \psi_i.$$ 

Using Clément interpolation operator, we now describe how we recover nodal gradients from $\nabla u_h \in \mathbb{P}^0$. For each $S_i \subset \mathcal{H}$ we have for the particular choice $1 \in \mathbb{P}^0$:

$$\int_{S_i} (\Pi_0(\nabla u_h) - \nabla u_h) = 0 \iff \int_{S_i} \Pi_0(\nabla u_h) = \int_{S_i} \nabla u_h$$

$$\iff |S_i| \Pi_0(\nabla u_h)|_{S_i} = \sum_{K \in S_i} \int_{K} \nabla u_h$$

$$\iff \Pi_0(\nabla u_h)|_{S_i} = \sum_{K \in S_i} |K| \nabla u_h|_{K}$$

where $|K|$ and $|S_i|$ denote the volume of element $K$ and stencil $S_i$, respectively. For each vertex $p_i$, we thus have the following gradient reconstruction:

$$\nabla R u_h(p_i) = \sum_{K \in S_i} |K| \nabla u_h|_{K}.$$ 

In fact, this procedure is equivalent to a reconstruction by means of a volume-weighted averaging.
The recovery procedure provides us with gradient nodal values and thus we get a piecewise linear representation of the gradient thanks to the Clément interpolation operator.

Similar relations are derived in three dimensions.

### 5.4.2 A double $L^2$-projection method

We can use the $L^2$-projection method to recover the Hessian of $u^*$ from $u_h$. To this end, the recovery described above is simply applied to each component of the gradient.

### 5.4.3 A method based on the Green formula

The Hessian of the solution can be recovered using a weak formulation based on the Green formula. We consider the same notations as previously. For each vertex $p_k$ of $H$, we have for $1 \leq i, j \leq 3$:

$$
\int_{S_k} \frac{\partial^2 u_h}{\partial x_i \partial x_j} \varphi_k = - \int_{S_k} \frac{\partial u_h}{\partial x_j} \frac{\partial \varphi_k}{\partial x_i} + \int_{\partial S_k} \frac{\partial u_h}{\partial n} \varphi_k d\sigma
$$

as the shape function is zero on the boundary of the stencil $\partial S_k$. A specific treatment is done close to the boundary. Each component of the Hessian is then recovered with the relation:

$$
\frac{\partial^2 u^*}{\partial x_i \partial x_j}(p_k) := - \int_{S_k} \frac{\partial u_h}{\partial x_j} \frac{\partial \varphi_k}{\partial x_i} - \sum_{K \in S_k} \frac{\partial \varphi_k}{\partial x_i} |S_k| \frac{1}{4}.
$$

which is equivalent to lump the mass matrix of the left-hand side of the previous relation.

### 5.4.4 A least-square approach

Let $p$ be a mesh vertex and let $u$ be the solution taken into account to define the metric. By considering a Taylor expansion of $u$ at a vertex $p_i$ connected to the vertex $p$ (i.e., $p_i \in B(p)$, the ball of $p$) and truncated at the order 2, we can write the following relation:

$$
u_i = u + pp_i \cdot \nabla u(p) + \frac{1}{2} \langle pp_i, H_u(p) pp_i \rangle
$$

$$
\Leftrightarrow \frac{1}{2} \langle pp_i, H_u(p) pp_i \rangle = u_i - u - pp_i \cdot \nabla u(p)
$$
with the notations \( u = u(p) \) and \( u_i = u(p_i) \). This relation can be developed using the notations:

\[
pp_i = (x_i, y_i, z_i), \quad \nabla u(p) = (\alpha, \beta, \gamma), \quad H_u(p) = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}
\]

as follows:

\[
\frac{1}{2}(ax_i^2 + 2bx_iy_i + 2cx_iz_i + dy_i^2 + 2ey_iz_i + fz_i^2) = u_i - u - (\alpha x_i + \beta y_i + \gamma z_i). \quad (5.3)
\]

This leads to a usually over-determined system\(^1\) of the form:

\[ AX = B, \quad \text{with} \quad ^tX = \begin{pmatrix} a & b & c & d & e & f \end{pmatrix} \]

where \( A \) is a \( n \times 6 \) matrix (\( n = \text{Card}(B(p)) \)) function of \((x_i, y_i, z_i)\) and \( B \) is a vector of dimension \( n \) given by the righthand side of the Relation (5.3), and function of \((\alpha, \beta, \gamma, x_i, y_i, z_i, u, u_i)\). This system is solved using a least-square approximation, i.e., it consists in minimizing the distance between the vectors \( AX \) and \( B \) of \( \mathbb{R}^6 \) by minimizing the square of the Euclidean norm of their difference. The problem is then to:

Find \( u \in \mathbb{R}^6 \) such that \( \|AX - B\|^2 = \inf_{Y \in \mathbb{R}^6} \|AY - B\|^2 \).

It can be shown that the solution of this problem is the solution of the linear \( 6 \times 6 \) system of normal equations:

\[ ^tAAX = ^tAB. \]

The latter is then solved using a standard Gauss method.

*Remark 5.4.1.* If, in some peculiar cases, the system is under-determined (i.e., \( \text{Card}(B(p)) < 6 \)), additional vertices connected to the vertices of \( B(p) \) can be taken into account.

\[5.5 Solution Interpolation\]

\[5.5.1 Localization algorithm\]

The localization problem or research of point location consists in identifying the element of a simplicial mesh containing a given point. The localization of a given point in a mesh is a frequent issue that arises in various situations. As regards interpolation methods, we initially have a mesh with a field, here the

\[1\] The system is over-determined as 6 coefficients must be computed and the vertex \( P \) is usually connected to more than 6 vertices \( p_i \) in three dimensions.
solution, that we call background mesh, denoted $H^{\text{back}}$. We aim at transferring or interpolating the field onto another mesh called current mesh or new mesh, denoted $H^{\text{new}}$. Therefore, the algorithm consists in finding which elements of the background mesh contain the vertices of the new mesh in order to apply an interpolation scheme.

Here, we consider the simplified problem where the background and the new meshes are discretizations of the same domain $\Omega$. This problem has to be dealt with care in the case of simplicial meshes to handle difficult configurations. Indeed, background and current meshes can be non-convex and can contain holes. It is also possible that the overlapping of the current mesh does not coincide with the background mesh since their boundary discretization can differ. Consequently, some vertices of the current mesh can be outside of the background mesh and conversely. Moreover, efficient localization algorithms have to be implemented to avoid the naive quadratic scheme in $O(N_{\text{ver}}^{\text{new}} \times N_{\text{tri}}^{\text{back}})$ where $N_{\text{ver}}^{\text{new}}$ is the number of vertices of $H^{\text{new}}$ and $N_{\text{tri}}^{\text{back}}$ the number of triangles of $H^{\text{back}}$.

The localization can be solved efficiently by traversing the background mesh using its topology, i.e., the neighboring elements of each element, thanks to a barycentric coordinates-based algorithm [31, 45]. More precisely, in two dimensions, let $p$ be a vertex of the new mesh, $K = [p_0, p_1, p_2]$ a triangle of the background mesh. From the signs of the three barycentric coordinates $\{\beta_i\}_{i=0,2}$, three possible cases arise (see Figure 5.4):

- all barycentric coordinates are positive then vertex $p$ is located inside element $K$
- one barycentric coordinate is negative then it indicates the direction for the next move. For instance, if barycentric $\beta_i$ is negative then we move to neighboring element $K_i$ sharing edge $e_i$ with $K$. We say that $p$ is viewed by edge $e_i$
- two barycentric coordinates are negative then two neighboring triangles are possible for the next move. A random choice or a geometric one is used.

Fig. 5.4. Illustration of the three possible cases depending on the signs of the three barycentric coordinates of vertex $p$ with respect to triangle $K$ when moving inside the background mesh.
Starting from an initial element $K_0$ of the background mesh, we apply the previous test. According to the signs of the barycentric coordinates, we pass through the corresponding neighbor of $K_0$ and we repeat this process until the three barycentric coordinates are positive meaning that the visited triangle contains $p$. With this algorithm, we follow a path in the background mesh to locate vertex $p$ as shown in Figure 5.5 (left). This algorithm complexity is in $O(n \times N_{\text{new}})$ where $n$ is the average number of visited triangles for each path.

However, cyclic or closed paths can occur. The element containing the vertex is missed and an infinite loop is obtained. In this case, the path leads us to an already tested element, as presented in Figure 5.5 (right). In this academic example, starting from $K_0$, triangles $K_1$, $K_2$, $K_3$, $K_4$ and $K_5$ are visited bringing us back to $K_0$. A color algorithm, to mark already visited elements, is used to avoid this problem allowing us to choose another direction when several choices reoccur. Another way to solve this problem is to consider a random choice when several possibilities occur.

Another difficulty arises when the path is stuck by the geometry of domain $\Omega$. Starting element $K_0$ and vertex $p$ are separated by a hole or by a non-convex domain, cf. Fig. 5.6 left. The path demands to pass through the hole or the boundary to reach the element containing vertex $p$. A simple, but inefficient, way to remedy this problem is to make an exhaustive search, i.e., for such a vertex all elements of the background mesh are tested. Besides, a more challenging solution is to follow a path on the boundary in order to bypass the obstacle.

**Localization coupled with a grid structure**

The previous algorithm can be very time consuming if a large number of elements (e.g. $n$ is large) needs to be visited between triangle $K_0$ and the solution triangle. This can result in a large number of area computations. This major drawback leads us to consider a more local approach which aims at combining the algorithm with a grid structure (a tree-like structure can be considered).

---

**Fig. 5.5.** Left, a possible path to locate the vertex $p$ of the new mesh starting from the triangle $K_0$ of the background mesh. Right, cyclic path leading us to an already checked element. Starting from $K_0$, triangles $K_1$, $K_2$, $K_3$, $K_4$ and $K_5$ are visited bringing us back to $K_0$. 
5.5 Solution Interpolation

This facilitates and speeds up the localization process. A grid enclosing the mesh is constructed and, for each grid cell, one element of the background mesh located in it, if any, is recorded. Then, to locate a new vertex in the background mesh, the cell containing the vertex is first identified and then the localization scheme starts from the element associated with this cell. In this way, the number of visited triangles is reduced and the number of necessary computations decreases as well. In the case where we are stuck by the boundary, because of a hole or a non-convex domain, the grid structure helps us to bypass the obstacle. Indeed, elements associated with grid neighboring cells of the current one are considered as new initial guesses for the searching algorithm. The localization is restarted from one of these new elements.

Remark 5.5.1. Note that the grid (or the tree-like structure) could be defined in various ways depending on the nature of the data set. In this respect, for a grid, the number of cells and thus the occupation of the cells are parameters that clearly affect the efficiency of the whole process.

Localization using the topology of both meshes

We can even improve the locality of the localization scheme by using the topology of both meshes. The previous algorithm can be very time consuming if a large number of elements (e.g. \( n \) is large) needs to be visited between triangle \( K_0 \) and the solution triangle. This major drawback leads us to consider a more local approach by using the topology of both meshes. Such scheme tends to minimize \( n \) the number of elements visited when locating new vertices. Instead of determining the location of the new vertices in their data (or storage) order, the idea is that once a vertex \( p \) has been located in a background element \( K \), then we handle the set of vertices \( \{ q_i \}_{i=1}^m \) of the ball of \( p \), i.e., the set of vertices that are connected to \( p \) by an edge. For the vertices \( \{ q_i \}_{i=1}^m \), we set as starting element of the localization process the triangle \( K \) that contains \( p \), see Figure 5.6 right. Consequently, the number of visited triangles is drastically reduced as in this algorithm the initial guess of the searching process is at the element (or connectivity) level. Moreover, with this approach, the scheme does not depend on any parameters.

Another advantage of this approach is that this scheme avoids the problem where the process is stuck by a hole or a non-convex boundary as vertices \( \{ q_i \}_{i=1}^m \) are connected to \( p \) in the new mesh. This algorithm is also in \( O(n \times N_{\text{new}}) \) where \( n \) is the average number of visited triangles and here \( n \) tends to be optimal. Indeed, in practice the number of visited triangles is on average less than 3. In fact \( n \) is of the order of the number of elements of the background mesh that are overlapped by an element of the current mesh.

Two problems still have to be handled with care: the "fork" and the boundary problems. The first one occurs when the path reaches a geometrical fork or a crossroads with multiple choices for the next move, then the presented algorithm could make the wrong choice and ask to process in the wrong direction. The second problem is that this algorithm is not able to handle the
case where vertices are located outside of the background discretized domain. Specific treatment are considered to solve such issues. At worst, it is always possible to perform an exhaustive search.

5.5.2 Classical polynomial interpolation

In this section, we present several interpolation algorithms which are not conservative. In this paper, the provided solution is considered to be piecewise linear by element. For such a scheme, the piecewise representation by element is assumed to be continuous. In the case of a nodal value representation of the solution, we get an implicit continuous piecewise linear solution by element. In the other cases, the solution representation must be modified to get piecewise linear continuous solution by element. Let us denote by $p$ be a vertex of the current mesh, $K = [p_0, p_1, p_2]$ a triangle of the background mesh containing $p$ and $\beta_i$, for $i = 0, ..., 2$, the barycentric coordinates of $p$ with respect to $K$. In the following, we denote by $P^k$ the set of polynomials of degree less or equal than $k$ and by $P_r$ the set of polynomials where lies the solution given by the interpolation scheme.

Linear interpolation

The easiest interpolation scheme is the classical $P^1$ interpolation, it reads:

$$u(p) = \sum_{i=0}^{2} \beta_i(p) u(p_i).$$

This scheme is $P^1$ exact, we have $P_r = P^1$ and it is of order 2. This scheme does not conserve the mass. Indeed, if an edge between two triangles is swapped

![Diagram](attachment:image.png)

**Fig. 5.6.** Left, starting element $K_0$ and vertex $p$ are separated by the non-convex domain. The path demands to pass through the boundary to reach the element containing vertex $p$. Right, the number of visited triangles in the localization scheme is reduced by using the topology of both meshes. Vertex $p$ has been located in element $K$. Then, the set of vertices $\{q_i\}_{i=1,m}$ connected to $p$ uses element $K$ as initial guess for the localization scheme.
then this interpolation keeps unchanged solution at the triangles vertices whereas the mass of the solution has changed.

**Quadratic interpolation**

The proposed quadratic scheme uses $P^2$ Lagrange test functions in triangle $K$ to reconstruct a quadratic representation of the solution on $K$. This interpolation requires the solution nodal value at triangle vertices $p_0$, $p_1$ and $p_2$, and the solution at the triangle mid-edges. We denote by $p_3$, $p_4$ and $p_5$ the middle of edges $e_0$, $e_1$ and $e_2$, respectively (i.e., $p_i$ is the middle of edge $p_{i-1}p_{i+1}$, for $i = 3, ..., 5$). The quadratic scheme is given by:

$$u(p) = \sum_{i=0}^{5} \psi_i(p) u(p_i)$$

with

$$\psi_i(p) = \beta_i(p) (2 \beta_i(p) - 1) \text{ for } i = 0, ..., 2,$$

$$\psi_i(p) = 4 \beta_i[p] \beta_{i+1}[p] \text{ for } i = 3, ..., 5.$$

If the mid-edge values are known then this interpolation is $P^2$ exact and hence of order 3. However, in our case the solution representation is continuous and piecewise linear by element. Therefore, we have to specify how the mid-edge solution values are obtained. From the solution nodal values, gradients at vertices could be reconstructed, see previous section. Let us denote by $\nabla u(p_i)$ the gradient for vertex $p_i$. In this case, we have an over-determined system for the quadratic reconstruction. Consequently, we choose to perform a cubic reconstruction on each edge to get the mid-edge values using the solution and the gradient at each extremity. After some algebra, mid-edge values are given by:

$$u(p_i) = \frac{u(p_{i-1}) + u(p_{i+1})}{2} + \frac{\nabla u(p_{i-1}) - \nabla u(p_{i+1})}{8} \cdot p_{i-1}p_{i+1}, \text{ for } i = 3, ..., 5.$$

The order and the exactness of the scheme depends on the gradient reconstruction. Nonetheless, for the proposed approach we have $P^1 \subset P_r \subset P^2$ and an order between 2 and 3 is obtained. The reconstruction is $C^1$ except through the elements edges. This interpolation is not conservative. The problem of the swap example holds. Moreover, this reconstruction does not verify the maximum principle. Hence, we have to limit this reconstruction in order to avoid the creation of new extrema. The proposed limiter is simply:

$$u_{lim}(p) = \max(\min(u(p), u_{max}), u_{min}),$$

with $u_{min} = \min_{i=0..2} u(p_i)$ and $u_{max} = \max_{i=0..2} u(p_i)$.

**5.6 Example of a CFD numerical simulation**

This CFD example concerns a classic numerical simulation of transonic air flow around the ONERA M6 wing. We assume that the flow is inviscid, it is thus
modeled by the compressible Euler equations. A second-order finite volume solver is used to compute the numerical solution [3]. The flow is computed for Mach number equal to 0.8395 with an angle of attack of 3.06 degrees. This transonic simulation case gives raise to a well-known lambda-shock.

The initial mesh is a relatively coarse mesh containing 7,815 vertices, 5,848 boundary triangles and 37,922 tetrahedra. The variable used to adapt the mesh is the Mach number. The mesh has been adapted 9 times, every 250 time steps. Figure 5.7 shows the adaptation in the isotropic and anisotropic cases. The final isotropic mesh contains 231,113 vertices and 1,316,631 tetrahedra and the final anisotropic mesh contains 23,516 and 132,676 tetrahedra. In this example, the maximal aspect ratio achieved for the anisotropic elements is about 10. Nevertheless, the anisotropic metric leads to a dramatic reduction of

Fig. 5.7. Onera M6 wing test case. Top, isotropic mesh adaptation and, bottom, anisotropic mesh adaptation. Left, adapted surface mesh and, right, cut through the volume adapted mesh.
the number of degrees of freedom, roughly one order less than in the isotropic case (for the same error level).

The CPU times required to generate the final surface (resp. volume) mesh is 31 (resp. 132) seconds, and to compute the Euler solution over 250 time steps is 2,782 seconds in the isotropic case, on a Pentium 4 3Mhz machine. Similarly, the CPU times required to generate the surface (resp. volume) mesh is 3 (resp. 25) seconds, and to compute the Euler solution over 250 time steps is 318 seconds in the anisotropic case.

The surface and volume meshes have been adapted using respectively the Yams [29] and Gamanic3d softwares [32].
References


