University of Padua

Department of Information Engineering

Department of Pure and Applied Mathematics

Course of Study in Computer Engineering

Approximation and Finite Elements in Isogeometric Problems

Graduand:
Luca Carlon

Supervisor:
Prof. Maria Morandi Cecchi

Academic year 2009/2010

Abstract The thesis begins with a short introduction to elliptic, parabolic and hyperbolic Partial Differential Equations and proceedes with a presentation of the Finite Element Method. Some important models for the representation of curves, surfaces and solids are explored before introducing a generalization of the Finite Element Method, namely Isogeometric Analysis. Algorithms for computation and numerical examples are presented.

Keywords Isogeometric Analysis, Finite Element Method, Partial Differential Equation

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

An efficient way of describing the behaviour of a system (e.g. natural systems) is to use a *mathematical model*. Among the mathematical models, *differential equations* (briefly described in Section 1.3) are commonly used when trying to relate functions to their rates of change. Examples of processes described by differential equations are constructions, weather, flow of liquids (e.g. flow of blood in human veins), deformation of solid bodies, heat transfer, chemical reactions, electromagnetism etc... Differential equations can be grouped in *Ordinary Differential Equations* (ODEs) and *Partial Differential Equations* (PDEs). ODEs are equations in which the unknown function is a function of a single variable; vice versa in PDEs the unknown function is a function of multiple variables. The manipulation of models is an important task for people to control, predict and understand the processes they describe.

Unfortunately, we are not able to find exact solutions for many of these types of equations, and sometimes we don't even know whether the solution exists or not or whether it is unique or not. However, approximation methods exist, and one of these is the *Finite Element Method* (FEM). FEM is currently one of the most efficient way of finding approximations for PDEs for which we're not able to find an exact solution. The idea of FEM is to divide the domain of the problem in *finite elements* where a finite set of polynomial *basis functions* is defined, and which create the basis for the space where the approximated solution is searched. This way, the problem is transformed into a discrete problem, as the solution is expressed through a finite number of unknown parameters. A key concept in this process is the choice of the finite set of polynomials used in the approximation: until recent times, mostly simple shape functions were used, so as to simplify implementation and usage. During the last years, the usage of higher-order shape functions has been reconsidered, due to their superior approximation capabilities. However, employing such more complex shape functions requires a better understanding of the underlying mathematics. Chapter 2 reports a brief discussion of finite elements and of the main concepts of FEM.

A new development of FEM has been recently proposed in [3, 18] (but already proposed before in [15]) and named *Isogeometric Analysis* (IGA). This can be considered an improvement which generalizes the concepts of FEM through a different definition of the basis functions using *CAD basis functions*, whose description is reported in Chapter 3.

This new idea is described in Chapter 4. Examples of usage of IGA are given

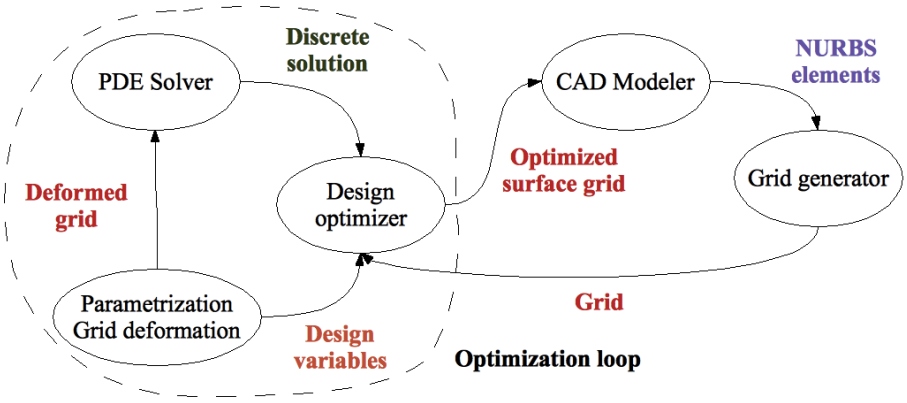

Figure 1.1: Current automated design loop.

in Chapter 5.

1.1 Automated Design Loop

Automated design loops are now used in industrial environments to create and optimize models. These loops are made up of many phases, each employing different input/output technologies and each belonging to different scientific fields. It is difficult, therefore, to let data be processed by all these phases: this results in a considerable overhead due to frequent translations and adaptations of the same model. A representation of the loop can be seen in Figure 1.1.

A *CAD modeler* is first used in order to design a first draft of the system under construction. This model can be described using any kind of *CAD functions*, such as *B-splines*, *NURBS's* or *T-splines* (see 3), for instance. This model is then the input of the *grid generator*, which creates a grid on the CAD model. This causes a deep modification of the geometry (see Figures 2.4, 2.3 and 5.9). Typically, in fact, piecewise-linear functions are used to approximate the boundaries, for example using a *triangulation* technique. The grid generated is then passed to the *optimization loop*, which simulates some kind of situation, and, according to the results, deforms the grid so as to maximize an objective function. The deformed grid is then passed back to the CAD modeler for the manufacturing process. This loop has several drawbacks:

- it spans many different technological fields, causing integration issues;
- it includes many different representations of similar concepts, requiring translations and conversions;
- it needs several different software with several different input/output formats.

The loop just presented is the loop resulting when FEM-based solvers are used.

A completely different approach is that of IGA (see Figure 1.2). The concept is to perform computations on parametric curves, surfaces and volumes instead of grids: thus, the grid generator is replaced by the *volumetric modeler*, which builds a CAD domain needed by the optimization loop. Simulations are, in the

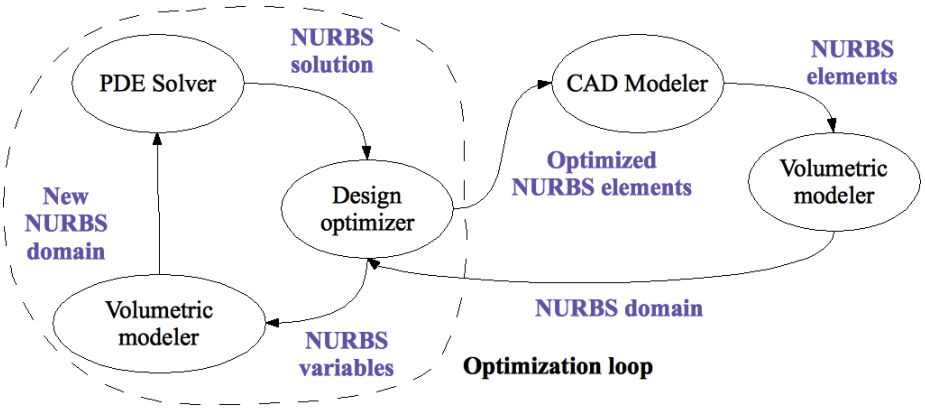

Figure 1.2: Isogeometric automated design loop.

loop, carried out performing integrations on the CAD basis functions. Far less overhead should be introduced in this approach, which can now be merged in one single process. This approach is a step towards integration of CAD and FEM.

1.2 Approximation

As already remarked, the problem of finding a function satisfying a PDE or a system of PDE and some additional constrains, is not exactly solvable in general. It is therefore necessary to approximate the resulting function starting from a certain number of known values, the *degrees of freedom* (DOFs). FEM and IGA are responsible of finding these values.

Assume C is a set of functions where $C \subset V$ and V is a linear space. If $g \in V$, an *approximation problem* consists in finding a function $g_C \in C$ which is close to g, where the term close is to be defined some way. The measure of the quality of the approximation can be defined, for instance, as the norm $\|g - g_C\|_V$ if the linear space V is a normed space. In this case, the best approximation is that function g_C which minimizes the distance between g_C and g. The approximation problem becomes an *interpolation* when g_C has to satisfy some constraints, generally defined by

$$L_i(g_C) = b_i, \ i = 1, \dots, N_C,$$

with $L_i : V \to \mathbb{R}$ independent linear forms in V' and b_i given constraints. This is the case for the Lagrange interpolation in 2.3.4.1, for instance.

It is clear that the choice of the space C is critical to get an approximation close to the exact solution. The features of the approximation using one of the sets or the other will be explored in detail in 2.2 and in 4.4.

1.2.1 Best interpolant

Assume V' is a finite-dimensional subset of the Hilbert space V. According to Lemma A.25, the closest element to $g \in V$ is the orthogonal projection

$g' = Pg \in V'$ in the norm $\|\cdot\|_V$. The orthogonal projection is defined as

$$\langle g - g', v' \rangle_V = 0, \ \forall v' \in V',$$

or, assuming $\{v'_1, \ldots, v'_N\} \subset V'$ is a basis of V'

$$\langle g - g', v'_i \rangle_V = 0, \ \forall v'_i, i = 1, \ldots, N. \tag{1.1}$$

Being V' a linear space and as $g' \in V'$ we can write it as the linear combination

$$g' = \sum_{j=1}^{N} \bar{g}'_j v'_j,$$

which can be substituted in (1.1), yielding

$$\sum_{j=1}^{N} \bar{g}'_j \langle v'_j, v'_i \rangle_V = \langle g, v'_i \rangle_V , \ \forall i = 1, \ldots, N.$$

This is a linear system in the unknowns $\bar{g}'_1, \ldots, \bar{g}'_N$.

1.3 Partial Differential Equations (PDEs)

PDEs encountered in physics and engineering are mostly second-order PDEs which are usually either *elliptic*, *parabolic* or *hyperbolic*. Elliptic equations describe the particular state of a system characterized by the minimum of a specific quantity (usually energy), parabolic problems mostly describe its evolution whereas hyperbolic equations model the transport of some physical quantity or information. Any other kind of second-order PDE is said to be undetermined.

The general form of a second-order PDE defined on an open connected[1] set (see Sub-subsection A.2.1) $Z \subset \mathbb{R}^n$ in n independent variables $\boldsymbol{z} = (z_1, \ldots, z_n)^T$ can be expressed as

$$-\sum_{i,j=1}^{n} \frac{\partial}{\partial z_i} \left(a_{ij} \frac{\partial u}{\partial z_j} \right) + \sum_{i=1}^{n} \left(\frac{\partial}{\partial z_i} (b_i u) + c_i \frac{\partial u}{\partial z_i} \right) + a_0 u = f \tag{1.2}$$

where a_{ij}, b_i, c_i, a_0 and f are all functions of the variable $\boldsymbol{z}$. For all derivatives to exist it is necessary to be: $u \in C^2(Z)$, $a_{ij} \in C^1(Z)$, $b_i \in C^1(Z)$, $c_i \in C^1(Z)$, $a_0 \in C(Z)$ and $f \in C(Z)$ (for the definition of continuity see Section B.1). According to this way of expressing a PDE, it is possible to define the terms elliptic, parabolic and hyperbolic used above.

Definition 1.1. According to the equation (1.2) and to the matrix $A(\boldsymbol{z}) = \{a_{ij}\}_{i,j=1}^{n}$, the equation is said to be elliptic at $\boldsymbol{z} \in Z$ if $A(\boldsymbol{z})$ is positive definite, parabolic if both at $\boldsymbol{z} \in Z$ $A(\boldsymbol{z})$ is positive semidefinite (and not definite) and the rank of $(A(\boldsymbol{z}), b(\boldsymbol{z}), c(\boldsymbol{z}))$ is n and hyperbolic when at $\boldsymbol{z} \in Z$ $A(\boldsymbol{z})$ has one negative and $n-1$ positive eigenvalues. An equation is elliptic, parabolic or hyperbolic when it is elliptic, parabolic or hyperbolic for all $\boldsymbol{z} \in Z$.

[1] A *connected space* is a topological space which cannot be represented as the union of two or more disjoint nonempty open subsets. A subset of a topological space X is a connected set if it is a connected space when viewed as a subspace of X.

Remark 1.2. In practice we distinguish between time-independent PDEs (like elliptic PDEs) and time-dependent PDEs. For a time-independent equation, we define $n = d$ (d is the spatial dimension) and $\boldsymbol{z} = \boldsymbol{x}$ ($\boldsymbol{x}$ is the spatial variable). In case the equation is time-dependant, it is useful to define $n = d+1$ (d spatial dimensions plus time) and $\boldsymbol{z} = (\boldsymbol{x}, t)$, where t represents the time. The same way, we define Z to be a domain which comprise both space and time, and we use Ω when the domain is time-independent. When there is the special case in which the equation is time-dependant but the spatial domain is not, we use the expression *space-time cylinder*, and we denote it with $Z = \Omega \times (0, T)$ (Ω is the spatial portion of the domain, $(0, T)$ is the time interval).

Definition 1.3. It is useful to define the *elliptic operator* L which allows the writing of PDEs in a compact form (elliptic equations are time-independent so $n = d$ and $u = u(\boldsymbol{x}) = u(\boldsymbol{z})$):

$$Lu = -\sum_{i,j=1}^{n} \frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial}{\partial x_j}\right) + \sum_{i=1}^{n}\left(\frac{\partial}{\partial x_i}(b_i u) + c_i\frac{\partial u}{\partial x_i}\right) + a_0 u. \tag{1.3}$$

1.4 Second-order elliptic equations

Second-order elliptic equations are briefly presented: weak formulation is derived, nonhomogeneous Dirichlet boundary conditions and Neumann boundary conditions are introduced.

1.4.1 Weak formulation

Assuming an elliptic equation in the form of the equation (1.2), it is possible to consider the alternate model equation

$$-\nabla \cdot (a_1 \nabla u) + a_0 u = f \tag{1.4}$$

where $a_{ij}(\boldsymbol{x}) = a_1(\boldsymbol{x})\delta_{ij}$ and $\boldsymbol{b}(\boldsymbol{x}) = \boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{0}$ in Ω, and Ω is $\Omega \subset \mathbb{R}^d$ with Lipschitz-continuous[2] boundary.

The simplest case is obtained imposing the *homogeneous Dirichlet boundary conditions*

$$u(\boldsymbol{x}) = 0, \ \forall \boldsymbol{x} \in \partial\Omega \tag{1.5}$$

which assure the solution function will vanish on the boundary of the domain.

In these conditions, the *strong* or *classical solution* to the problem consisting of the equations (1.4) and (1.5) is a function $u \in C^2(\Omega) \cap C\left(\overline{\Omega}\right)$ for which (1.4) is true for all $\boldsymbol{x} \in \Omega$ and (1.5) is true for all $\boldsymbol{x} \in \partial\Omega$. Unfortunately, it is not possible to guarantee the solvability of the problem in any way.

Nevertheless, it is possible to introduce an alternative model of the problem (1.4), (1.5). This model, named *weak formulation*, can be derived following four steps which begin with the use of (1.4).

[2] Given two metric spaces (X, d_X) and (Y, d_Y), where d_X denotes the metric on the set X and d_Y denoted the metric on the set Y, a function $f : X \to Y$ is called *Lipschitz-continuous* if there exists a real constant $K \geq 0$ such that, for all x_1 and x_2 in X

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2).$$

1. Multiply with a *test function* (also named *weight function* of *variation*) $\varphi \in C_0^\infty(\Omega)$ (for a definition of $C_0^\infty(\Omega)$ see Sub-subsection A.4.1):

$$-\nabla \cdot (a_1 \nabla u)\, \varphi + a_0 u \varphi = f\varphi.$$

2. Integrate over the domain Ω:

$$-\iint_\Omega \nabla \cdot (a_1 \nabla u)\, \varphi d\boldsymbol{x} + \iint_\Omega a_0 u \varphi \boldsymbol{x} = \iint_\Omega f\varphi \boldsymbol{x}.$$

3. Using the Green's theorem (see Subsection A.4.2) it is possible to reduce the maximum order of the derivatives:

$$\iint_\Omega a_1 \nabla u \cdot \nabla \varphi d\boldsymbol{x} + \iint_\Omega a_0 u \varphi d\boldsymbol{x} = \iint_\Omega f\varphi d\boldsymbol{x} + \oint_{\partial\Omega} \varphi a_1 \frac{\partial u}{\partial \boldsymbol{n}} d\boldsymbol{S}.$$

   Furthermore, it is to consider that φ is a distribution and, for what is explained in Sub-subsection A.4.1, is null on the boundary of the domain Ω, leading to

$$\iint_\Omega a_1 \nabla u \cdot \nabla \varphi d\boldsymbol{x} + \iint_\Omega a_0 u \varphi d\boldsymbol{x} = \iint_\Omega f\varphi d\boldsymbol{x}. \tag{1.6}$$

4. It is possible to change the spaces assumed above, relaxing the restrictions. It was necessary to assume $u \in C^2(\Omega) \cap C\left(\overline{\Omega}\right)$ and $\varphi \in C_0^\infty(\Omega)$ but, considering the equation (1.6) it is sufficient to assume $u \in U$ and $\varphi \in V$ with $U = V = H_0^1(\Omega)$, $f \in L^2(\Omega)$ and $a_1, a_0 \in L^\infty(\Omega)$.

It is therefore possible now to formulate the problem in a different way: given $f \in L^2(\Omega)$, find a $u \in U$ so that

$$\iint_\Omega (a_1 \nabla u \cdot \nabla \varphi + a_0 u \varphi)\, d\boldsymbol{x} = \iint_\Omega f\varphi d\boldsymbol{x},\ \forall \varphi \in V.$$

The writing of the formulae above can be simplified employing the bilinear form $a(\cdot,\cdot) : U \times V \to \mathbb{R}$ and a linear form $l(\cdot) : V \to \mathbb{R}$ defined respectively:

$$a(u,\varphi) = \iint_\Omega (a_1 \nabla u \cdot \nabla \varphi + a_0 u \varphi)\, d\boldsymbol{x}$$

and

$$l(\varphi) = \iint_\Omega f\varphi d\boldsymbol{x}.$$

(1.6) can then be rewritten in the simpler way

$$a(u,\varphi) = l(\varphi).$$

1.4.2 Nonhomogeneous Dirichlet boundary conditions

The model reported so far assumes the homogeneous Dirichlet boundary conditions of the equation (1.5). It is necessary to extend the model in order to accommodate for the presence of the nonhomogeneous Dirichlet boundary conditions

$$u(\boldsymbol{x}) = g(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \partial\Omega. \tag{1.7}$$

where $g \in C(\partial\Omega)$. This requires to redefine the set of trial solutions U with

$$U = \left\{ u | u \in H^1(\Omega), u(\boldsymbol{x}) = g(\boldsymbol{x}) \, \forall \boldsymbol{x} \in \partial\Omega \right\}.$$

For reasons that will be explained afterwards, a new function $\gamma \in C^2(\Omega) \cap C\left(\bar{\Omega}\right)$ (called Dirichlet lift) for which $\gamma = g$ on the boundary of Ω so that

$$u = \gamma + v, \tag{1.8}$$

is defined, with $v \in V$. By substitution, the problem (1.4), (1.7) yields:

$$\begin{cases} -\nabla \cdot (a_1 \nabla v) + a_0 v = f + \nabla \cdot (a_1 \nabla \gamma) - a_0 \gamma, & \forall \boldsymbol{x} \in \Omega \\ v = 0, & \forall \boldsymbol{x} \in \partial\Omega \end{cases}.$$

Following the four steps of the sub-subsection 1.4.1, the weak formulation reads: given $f \in L^2(\Omega)$, find $v \in V$ for which

$$\iint_\Omega (a_1 \nabla v \cdot \nabla \varphi + a_0 v \varphi) \, d\boldsymbol{x} = \iint_\Omega (f\varphi - a_1 \nabla \gamma \cdot \nabla \varphi - a_0 \gamma \varphi) \, d\boldsymbol{x}, \ \forall \varphi \in V$$

or in the language of the linear forms

$$a(v, \varphi) = l(\varphi), \ \forall \varphi \in V,$$

with

$$a(v, \varphi) = \iint_\Omega (a_1 \nabla v \cdot \nabla \varphi + a_0 v \varphi) \, d\boldsymbol{x}, \ \varphi \in V$$

$$l(\varphi) = \iint_\Omega (f\varphi - a_1 \nabla \gamma \cdot \nabla \varphi - a_0 \gamma \varphi) \, d\boldsymbol{x}, \ \varphi \in V$$

given (1.7) and (1.8).

It is possible to prove that the solution is independent on the Dirichlet lift γ.

1.4.3 Neumann boundary conditions

There are different kinds of boundary conditions which can be imposed to the problem of the equation (1.4). Neumann boundary conditions can be expressed with

$$\frac{\partial u(\boldsymbol{x})}{\partial \boldsymbol{\nu}} = g(\boldsymbol{x}), \ \forall \boldsymbol{x} \in \partial\Omega. \tag{1.9}$$

with $g \in C(\partial\Omega)$.

Following the usual four steps for the formulation of the weak form of the problem, it can be seen that the function φ does not vanish anymore on the boundary. This is because we now have to require both u and v to be $C^\infty(\Omega) \cap C^1(\Omega)$, as they have to be differentiable on the boundary of Ω, so that v doesn't vanish anymore and a new integral appears in the final formulation, which becomes: given $f \in L^2(\Omega)$ and $g \in L^2(\partial\Omega)$, find $u \in U = H^1(\Omega)$ (solutions are not required to vanish on the boundary anymore, so the trial solutions space is no more H_0^1 but only H^1) such that

$$\iint_\Omega (a_1 \nabla u \cdot \nabla \varphi + a_0 u \varphi) \, d\boldsymbol{x} = \iint_\Omega f\varphi d\boldsymbol{x} + \oint_{\partial\Omega} a_1 g \varphi d\boldsymbol{S}, \ \forall \varphi \in V = H^1(\Omega)$$

or in the language of the linear forms

$$a(u,\varphi) = l(\varphi),\ \forall\varphi \in H^1(\Omega)$$

where

$$a(u,\varphi) = \iint_\Omega (a_1 \nabla u \cdot \nabla\varphi + a_0 u\varphi)\, d\boldsymbol{x},\ \forall\varphi, u \in H^1(\Omega)$$

$$l(\varphi) = \iint_\Omega f\varphi d\boldsymbol{x} + \oint_{\partial\Omega} a_1 g\varphi d\boldsymbol{S},\ \forall\varphi \in H^1(\Omega),$$

given (1.9).

1.4.4 Combination of boundary conditions

A formulation for a problem which is characterized by both Dirichlet and Neumann boundary conditions can be derived from the above discussion. The boundary $\partial\Omega$ is divided in two parts such that $\partial\Omega = \Gamma_N \cup \Gamma_D$, where Γ_N is the part of the boundary on which Neumann boundary conditions are imposed and where Γ_D is the part on which Dirichlet conditions are defined. The original form of the problem is then:

$$\begin{cases} -\nabla\cdot(a_1\nabla u) + a_0 u = f, & \forall\boldsymbol{x}\in\Omega \\ u = g_D, & \forall\boldsymbol{x}\in\Gamma_D \\ u = g_N, & \forall\boldsymbol{x}\in\Gamma_N \end{cases}. \tag{1.10}$$

Before beginning with the first step of the process of derivation of the weak formulation, it is necessary to extend the function g_D which needs to be $C(\Gamma_D)$ with a function $\tilde{g}_D \in C(\partial\Omega)$ such that it is $\tilde{g}_D \equiv g_D$ on Γ_D. Next, as stated above, it is necessary to define a Dirichlet lift γ so that it is possible to express the solution u with the sum $\gamma + \upsilon$ with $\gamma = \tilde{g}_D$. By substitution we get

$$\begin{cases} -\nabla\cdot(a_1\nabla(\gamma+\upsilon)) + a_0(\gamma+\upsilon) = f, & \forall\boldsymbol{x}\in\Omega \\ \gamma + \upsilon = g_D, & \forall\boldsymbol{x}\in\Gamma_D \\ \dfrac{\partial(\gamma+\upsilon)}{\partial\boldsymbol{\nu}} = g_N, & \forall\boldsymbol{x}\in\Gamma_N \end{cases},$$

which can be further modified to get

$$\begin{cases} -\nabla\cdot(a_1\nabla\upsilon) + a_0\upsilon = f + \nabla\cdot(a_1\nabla\gamma) - a_0\gamma, & \forall\boldsymbol{x}\in\Omega \\ \upsilon = 0, & \forall\boldsymbol{x}\in\Gamma_D \\ \dfrac{\partial(\gamma+\upsilon)}{\partial\boldsymbol{\nu}} = g_N, & \forall\boldsymbol{x}\in\Gamma_N \end{cases}. \tag{1.11}$$

The four steps for the derivation of the weak formulation yield to the problem: find a function $\upsilon \in H^1(\Omega)$ such that

$$\iint_\Omega (a_1\nabla\upsilon\cdot\nabla\varphi + a_0\upsilon\varphi)\, d\boldsymbol{x} =$$

$$\iint_\Omega (f\varphi - a_1\nabla\gamma\cdot\nabla\varphi - a_0\gamma\varphi)\, d\boldsymbol{x} + \int_{\Gamma_N} (a_1 g_N\varphi)\, d\boldsymbol{S},\ \forall\varphi\in H^1(\Omega). \tag{1.12}$$

The result can now be rewritten in the language of the linear forms:

$$a(v,\varphi) = l(\varphi),\ \forall\varphi \in H^1(\Omega) \tag{1.13}$$

where

$$a(v,\varphi) = \iint_\Omega (a_1 \nabla v \cdot \nabla\varphi + a_0 v\varphi)\, d\boldsymbol{x},\ v,\varphi \in H^1(\Omega)$$

$$l(\varphi) = \iint_\Omega (f\varphi - a_1 \nabla\gamma \cdot \nabla\varphi - a_0\gamma\varphi)\, d\boldsymbol{x} + \int_{\Gamma_N} (a_1 g_N \varphi)\, d\boldsymbol{S},\ \forall\varphi \in H^1(\Omega).$$

1.5 Second-order parabolic equations

Another important class of PDEs is the class of second-order parabolic equations, which is a generalization of an important particular case: the *heat equation*. Let $\Omega \subset \mathbb{R}^d$ be an open set with Lipschitz-continuous boundary. We will refer to the equation in the form

$$\frac{\partial u(\boldsymbol{x},t)}{\partial t} + L(u(\boldsymbol{x},t)) = f(\boldsymbol{x},t),\ \text{in } \Omega, \tag{1.14}$$

where t is the temporal variable, $n = d+1$ and L is an elliptic operator with time-independent coefficients only of the form of Equation (1.3). Equation (1.14) is considered in the time cylinder introduced in Remark 1.2 $Z_T = \Omega \times (0,T)$, with $T > 0$.

1.5.1 Weak formulation

The simplest case is that of homogeneous Dirichlet boundary condition, which yields a problem of the form

$$\begin{cases} \dfrac{\partial u}{\partial t} + Lu = f,\ \text{in } Z_T \\ u = 0,\ \text{on } \partial\Omega \times (0,T) \\ u = g,\ \text{on } \Omega \times \{t = 0\} \end{cases}.$$

It is necessary to assume $a_{ij}, b_i, c_i \in L^\infty(Z_T)$, $f \in L^2(Z_T)$ and $g \in L^2(Z_T)$. The process of derivation of the weak formulation is similar to that used in the elliptic case, except for the presence of the time-dependant part of the equation. For this we need to define some spaces.

Definition 1.4. By $L^q\left(0,T;W^{k,p}(\Omega)\right)$ we denote the space

$$\left\{ u : (0,T) \to W^{k,p}(\Omega),\ u \text{ is measurable and } \int_0^T \|u(t)\|_{k,p,\Omega}^q\, dt < \infty \right\},$$

endowed with the norm

$$\|u\|_{L^q(0,T;W^{k,p}(\Omega))} = \left(\int_0^T \|u(t)\|_{k,p,\Omega}^q\, dt \right)^{\frac{1}{q}}.$$

The symbol $u(t)$ indicates a function of $\boldsymbol{x}$ such that $u(t) : \boldsymbol{x} \to u(\boldsymbol{x},t)$.

Definition 1.5. We define the space

$$C\left([0,T];L^p(\Omega)\right)=\left\{u:[0,T]\to L^p(\Omega):\|u(t)\|_{p,\Omega}\ \text{is continuous in}\ [0,T]\right\}.$$

Analogously we define the space

$$C\left([0,T];W^{k,p}(\Omega)\right)=\left\{u:[0,T]\to W^{k,p}(\Omega):\right.$$

$$\left.\|u(t)\|_{k,p,\Omega}\ \text{is continuous in}\ [0,T]\right\}.$$

With these definitions it is possible to derive a weak formulation for homogeneous Dirichlet boundary conditions. The problems with the assumptions above and $u_0\in H_0^1(\Omega)$ is to find $u\in L^2(0,T;V)\cap C\left([0,T];L^2(\Omega)\right)$ for which

$$\frac{d}{dt}\langle u(t),v\rangle_{L^2}+a(u(t),v)=\langle f(t),v\rangle_{L^2},\ \forall v\in V,\ t\in(0,T)$$

and

$$u(0)=u_0.$$

The definition of the bilinear form $a(\cdot,\cdot)$ is

$$a(u,v)=\int_\Omega\left[\sum_{i,j=1}^{d}a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i}-\sum_{i=1}^{d}\left(b_iu\frac{\partial v}{\partial x_i}-c_iv\frac{\partial u}{\partial x_i}\right)+a_0uv\right]d\boldsymbol{x}. \quad (1.15)$$

The nonhomogeneous Dirichlet boundary conditions are treated the same way they were treated in the elliptic case, using the Dirichlet lift; the same holds for the Neumann boundary conditions.

1.6 Second-order hyperbolic equations

The form of an hyperbolic equation can be expressed with the equation

$$\frac{\partial^2u}{\partial t^2}+Lu=f,$$

where L is en elliptic operator of the form

$$L=\sum_{i,j=1}^{d}\frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial}{\partial x_j}\right).$$

As in the case of parabolic equations we are interested in solving the equation in the time cylinder $Z_T=\Omega\times(0,T)$, with $\Omega\subset\mathbb{R}^d$ an open bounded set with Lipschitz-continuous boundary.

1.6.1 Weak formulation

Given the definition of the problem and the definitions of Sub-subsection 1.5.1 we can formulate the problem: given $f\in L^2(Z_T)$ and the initial conditions $u_0\in$

$H_0^1(\Omega)$ and $u_1 \in L^2(\Omega)$, find a function $u \in C\left([0,T];H_0^1(\Omega)\right) \cap C^1\left([0,T];L^2(\Omega)\right)$ such that

$$\begin{cases} \dfrac{d^2}{dt^2}\langle u(t), v\rangle_{L^2} + a(u(t), v) = \langle f(t), v\rangle_{L^2}, \ \forall v \in H_0^1(\Omega), \ t \in (0,T) \\ u(0) = u_0 \\ \dfrac{du}{dt}(0) = u_1 \end{cases}$$

where the definition of $a(\cdot,\cdot)$ is that given in Equation (1.15).

Chapter 2

Finite Element Method

A very efficient and widely used method for approximating solutions of PDEs is the Finite Element Method. In this chapter the fundamental elements are presented. In Section 2.1 we describe the idea of the Galerkin method, which is necessary to obtain a discrete problem, so that a solution $u \in V$, where V is a space of infinite dimension, is approximated with a function $u_n \in V_n$ with V_n of finite dimension. When the space V_n comprise piecewise-polynomial functions only, the Galerkin method is called Finite Element Method. FEM divides the domain of the problem in finite elements, and defines the functions belonging to V_n on them. In Section 2.2, the finite elements are formally defined. In 2.3 one-dimensional problems are examined, and the order of the polynomials is discussed. Two-dimensional and three-dimensional problems are reported in 2.4 and 2.5. In 2.6 the description of the geometry is taken into account, and in 2.7 integration over this geometry is analyzed. Some discussions on performance and efficiency are then proposed.

2.1 Galerkin method

The Galerkin method faces the problem of finding a function u belonging to a Hilbert space V such that

$$a(u, \varphi) = l(\varphi), \ \forall \varphi \in V \tag{2.1}$$

with $a(\cdot, \cdot) : V \times V \to \mathbb{R}$ and $l(\cdot) : V \to \mathbb{R}$. The problem expressed by the equation (2.1) is formulated in a space V of infinite dimensions. It is not possible therefore to find a solution as a function of an infinite number of unknown parameters. Hence, the idea of the method is to reduce the dimension of the space V, defining a sequence of subspaces such that

$$V_n \subset V_{n+1} \subset \ldots \subset V. \tag{2.2}$$

In each subspace it is possible to find an approximate solution of the problem of the equation (2.1). If we consider the sequence of solutions $u_n \in V_n, \ n = 1, \ldots, +\infty$ it can be seen that they converge to the exact solution $u \in V$

$$\lim_{n \to +\infty} \|u_n - u\| = 0.$$

The problem of finding a solution $u_n \in V_n$ satisfying

$$a(u_n, \varphi) = l(\varphi), \ \forall \varphi \in V_n \tag{2.3}$$

is called *discrete problem*.

The solution to the problem (2.3) can be found considering that u_n is a function belonging to a space of finite dimension which is also a linear space $\dim(V_n) = N_n$, and so it can be expressed as a linear combination of the basis functions:

$$u_n = \sum_{i=0}^{N_n-1} \bar{u}_i v_i$$

where $\{v_i\}_{i=0}^{N_n-1}$ is a basis for the space V_n. By substitution in (2.3) we get

$$a\left(\sum_{i=0}^{N_n-1} \bar{u}_i v_i, \varphi\right) = l(\varphi), \ \forall \varphi \in V_n.$$

Considering the linearity of the $a(\cdot,\cdot)$ operator it is possible to move the coefficients of the linear combination outside of the operator:

$$\sum_{i=0}^{N_n-1} \bar{u}_i a(v_i, \varphi) = l(\varphi), \ \forall \varphi \in V_n.$$

The equation needs to be an identity for all the functions $\varphi \in V_n$ and so even for the basis functions $\{v_j\}_{j=0}^{N_n-1}$. We obtain:

$$\sum_{i=0}^{N_n-1} \bar{u}_i a(v_i, v_j) = l(v_j), \ j = 0, \ldots, N_n - 1. \tag{2.4}$$

It is possible to write equation (2.4) using matrices:

$$\boldsymbol{S}_n \cdot \bar{\boldsymbol{U}}_n = \boldsymbol{F}_n,$$

where

$$\boldsymbol{S}_n = [a(v_i, v_j)]_{i,j=0}^{N_n-1}$$

is the *stiffness matrix*,

$$\bar{\boldsymbol{U}}_n = [\bar{u}_i]_{i=0}^{N_n-1}$$

is the *unknown vector* and

$$\boldsymbol{F}_n = [l(v_j)]_{i=0}^{N_n-1}$$

is the *force vector*.

Remark 2.1. In this presentation the subscript n has been used to distinguish between Galerkin subspaces. However, the subscript h is often used, and it indicates the size of the elements of the mesh[1], as we will see afterwards in Subsection 2.3.1. Nothing changes in what has been said, as h can be seen as a function $h(n)$ so that in the Galerkin procedure

$$\lim_{n \to +\infty} u_n = \lim_{h(n) \to 0} u_{h(n)}.$$

[1] The *mesh* is a subdivision of the domain. It will be explained in more details afterwards.

2.1.1 Convergence

Theorem 2.2. *Let V be an Hilbert space and $V_1 \subset V_2 \subset \ldots \subset V$ a sequence of its finite dimensional subspaces such that*

$$\overline{\bigcup_{n=1}^{+\infty} V_n} = V.$$

Let $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ be a bounded bilinear V-elliptic form and $l \in V'$. Then

$$\lim_{n\to+\infty} \|u - u_n\|_V = 0. \tag{2.5}$$

Remark 2.3. By saying that (2.5) we say that the Galerkin method converges to the exact solution when approximating with spaces nearer and nearer to the original space V.

2.2 Nodal elements and unisolvency

A fundamental concept in finite element analysis is that of the nodal element. A possible formal definition can be given:

Definition 2.4. The *nodal finite element* is a triple $(K, P, \mathcal{L})$ where

- $K \subset \mathbb{R}^d$ is a bounded domain with Lipschitz-continuous boundary;
- P is a space of polynomials defined on the domain K with $\dim(P) = N_P$;
- $\mathcal{L} = \{L_0, L_1, \ldots, L_{N_P-1}\}$ is a set of linear forms (called *degrees of freedom* (DOF)) whose definition is

  $$L_j : g \in P \to g(\boldsymbol{x}_j) \in \mathbb{R}$$

  where the $\boldsymbol{x}_j$'s are the nodal points.

Given the definition of nodal finite element, it is important to define the concept of *unisolvency*: this is a property a nodal finite element is required to have so that it is guaranteed that the vector

$$\bar{\boldsymbol{U}}_n = \boldsymbol{S}_n^{-1} \cdot \boldsymbol{F}_n$$

identifies a unique polynomial

$$u_n = \sum_{i=0}^{n-1} \bar{u}_i v_i$$

where u_n is a solution of the discrete problem and $v_0, v_1, \ldots, v_{n-1}$ is a basis of the space where to find the solution.

Definition 2.5. A nodal finite element $(K, P, \mathcal{L})$ is said to be unisolvent if for every $g \in P$

$$L_0(g) = L_1(g) = \ldots = L_{N_P-1}(g) = 0 \Rightarrow g = 0.$$

Lemma 2.6. *Assuming $(K, P, \mathcal{L})$ is a unisolvent nodal finite element, given any set of numbers $\{g_0, g_1, \ldots, g_{N_P-1}\} \in \mathbb{R}^{N_P}$, with $dim(P) = N_P$, there exists a unique polynomial $g \in P$ so that*

$$L_0(g) = g_0,\ L_1(g) = g_1,\ \ldots, L_{N_P-1}(g) = g_{N_P-1}.$$

Definition 2.7. Assuming $(K, P, \mathcal{L})$ is a nodal finite element where $\dim(P) = N_P$, a set of functions $\mathcal{P} = \{v_0, v_1, \ldots, v_{N_P-1}\}$ is a *nodal basis* of P if it is true that

$$L_i(v_j) = \delta_{ij},\ \forall 0 \leq i, j \leq N_P - 1. \tag{2.6}$$

In this case the functions v_i are called *nodal shape functions.*

Given these definitions it is possible to enunciate the theorem on unisolvency.

Theorem 2.8. *Consider a nodal finite element $(K, P, \mathcal{L})$ with $dim(P) = N_P$. The finite element is unisolvent if and only if there exists a unique nodal basis $\mathcal{P} = \{v_0, v_1, \ldots, v_{N_P-1}\} \subset P$.*

Theorem 2.8 allows the definition of a procedure for checking the unisolvency of a defined nodal finite element.

1. Define the arbitrary basis $\{v_0, v_1, \ldots, v_{N_P-1}\}$.
2. Build the Vandermonde matrix $\boldsymbol{L} = [L_i(v_j)]_{i,j=0}^{N_P-1}$.
3. If $\boldsymbol{L}$ is invertible the element is unisolvent, otherwise it's not. If $\boldsymbol{L}^{-1}$ exists, then it has the coefficients $a_{jk},\ j = 0, 1, \ldots, N_P - 1$ in its k^{th} column.

Another important definition is that of the local nodal interpolant. The interpolation on finite elements is a procedure that takes a function $g \in V(\Omega_h)$ and produces a suitable piecewise-polynomial representant in the finite element space $g_{h,p} \in V_{h,p}(\Omega_h)$, where Ω_h is an approximation of the domain Ω, which can even be exact, and $V_{h,p}$ is the space of piecewise-polynomials which will be defined more precisely in Section 2.4.

Definition 2.9. Let $\{v_0, \ldots, v_{N_P-1}\}$ be the unique nodal basis for the unisolvent finite element $(K, P, \mathcal{L})$. Let $g \in V$, where $P \subset V$, be a function for which the values $L_0(g), \ldots, L_{N_P-1}(g)$ are defined. Then the *local nodal interpolant* is defined as

$$\mathcal{I}_K(g) = \sum_{i=0}^{N_P-1} L_i(g)\, v_i.$$

Interpolation on the finite elements will be even clearer when Theorem 2.24 will be enunciated.

2.2.1 Reference maps and isoparametric concept

An important concept in FEM is that of the *affine reference map* and of the *reference domain*, as it makes the formulation and the implementation of FEM systems more simple and efficient. The idea is to work on a reference domain using a suitable set of nodal shape functions defined on it, reducing every other nodal finite element to the reference by way of an affine reference map. More

precisely, we can define a reference domain $\tilde{K}$, a set of nodal shape functions on $\tilde{K}$ where N_{en} is the number of nodes of the element and a reference map

$$\boldsymbol{x}_{K_m} : \tilde{K} \to K_m$$

so that it is possible to map the nodal shape functions from the reference domain to the domain K_m without the need to define nodal shape functions on every domain of the mesh.

A reference map makes us able to map generally arbitrarily curved elements of the mesh to reference elements. If the edges or the faces of the elements are parametrized by non-polynomial functions, then the reference map is non-polynomial. For an extensive study on the construction of non-polynomial reference maps see [31].

In order to make the implementation more efficient and simple, it is possible to consider an approximation of reference maps called *isoparametric*. Isoparametric reference maps are easier to store and to handle, partial and inverse derivatives can be calculated more efficiently. Reference maps are, so, approximated by polynomial functions that, for each element $\mathcal{K} \in \mathcal{M}_{h,p}$, are defined as a linear combination of the shape functions of the reference domain with vector-valued coefficients. The isoparametric concept is commonly attributed to [33, 20] and is based upon the use of the same shape functions for the definition of the reference maps and for the approximate solution of the problem. It has to be noticed anyway that no relation between reference maps and approximate solution exists. For lowest-order elements:

Definition 2.10. Let $\boldsymbol{x}_{K_m} : \tilde{K} \to K_m$ be of the form

$$\boldsymbol{x}(\boldsymbol{\xi}) = \sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \boldsymbol{x}_i^{(m)}$$

where $\boldsymbol{x}_i$ is the i^{th} node of the element K_m and $v_{\tilde{K}}^{\boldsymbol{\xi}_i}$ is the i^{th} nodal shape function. If the element interpolation function $u_{h,p}$ can be written as

$$u_{h,p}(\boldsymbol{\xi}) = \sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \bar{u}_i^{(m)} \tag{2.7}$$

where $\bar{u}_i^{(m)}$ is the i^{th} degree-of-freedom of the element K_m, the element is said to be *isoparametric*.

2.2.2 Nodal elements and convergence of the Galerkin method

A fundamental condition when building nodal elements is that it is desirable (and in practice necessary) that nodal elements are designed in a way such that as the mesh is refined, the solution of the Galerkin method converges to the exact solution. This happens when the refinement generates a situation like the one reported in 2.1.1.

It is possible to enunciate three conditions, which are only sufficient and not necessary, for the convergence of the Galerkin method. Shape functions need to be:

Condition 2.11. smooth (C^1 shape functions) on the element interior;

Condition 2.12. continuous on the boundary of each element;

Condition 2.13. complete.

Condition 2.11 and 2.12 guarantee that the first derivatives which are to be computed exist. Guaranteeing C^0 at the boundaries means we will have, at worst, finite jumps in the first derivatives. If we permitted discontinuities, we would have deltas in the derivatives and squares of the delta function would appear in the calculation of the stiffness integral. The completeness property requires, instead, that the element interpolation function is capable of exactly representing an arbitrary linear polynomial (when considering lowest-order elements) when the nodal degrees-of-freedom are given values in accordance with it. Completeness is reasonable when thinking that, as the mesh is refined, the exact solution approaches constant values over an element. This property assures that constant and linear functions can be represented.

2.2.3 Convergence of isoparametric elements

Definition 2.14. A mapping $\boldsymbol{x}_{K_m} : \tilde{K} \to K_m \subset \mathbb{R}^d$ is said to be *one-to-one* if for each pair of points $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2 \in \tilde{K}$ such that $\boldsymbol{\xi}_1 \neq \boldsymbol{\xi}_2$, then $\boldsymbol{x}(\boldsymbol{\xi}_1) \neq \boldsymbol{x}(\boldsymbol{\xi}_2)$.

Definition 2.15. A mapping $\boldsymbol{x}_{K_m} : \tilde{K} \to K_m \subset \mathbb{R}^d$ is said to be *onto* if $\tilde{K} = \boldsymbol{x}\left(\tilde{K}\right)$.

As a consequence of the inverse function theorem, if $\boldsymbol{x}_{K_m}$ is:

Condition 2.16. one-to-one;

Condition 2.17. onto;

Condition 2.18. C^k, $k \geq 1$;

Condition 2.19. $J_{\boldsymbol{x}_{K_m}}(\boldsymbol{\xi}) > 0$, $\forall \boldsymbol{\xi} \in \tilde{K}$,

then the inverse mapping $\boldsymbol{\xi}_{K_m} = \boldsymbol{x}_{K_m}^{-1} : K_m \to \tilde{K}$ exists and is C^k.

Proposition 2.20. *Let the reference map satisfy all the conditions from 2.16 to 2.19. Then the smoothness requirement of point 1 is satisfied as well.*

Remark 2.21. In practice, the mappings satisfy all the points from 2.16 to 2.19, with the exception of mappings defining degenerations with coalesced nodes, like the case of the triangular domain, which is seen as a quadrilateral with a edge of length 0. In this case Condition 2.19 is not satisfied, in fact the Jacobian determinant vanishes at certain nodal points. With the exception of these points however, the mapping $\boldsymbol{\xi}_{K_m}$ remains smooth.

The third convergence condition is verified if the following proposition holds.

Proposition 2.22. *If* $\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} = 1$*, then the completeness condition is satisfied for isoparametric elements.*

Proof. Supposing $d = 3$ (the same holds for other values) and lowest-order elements

$$\begin{aligned}
\upsilon_{h,p}(\boldsymbol{\xi}) &= \sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \bar{v}_i, \\
&= \sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \left(c_0 + c_1 x_i^{(m)} + c_2 y_i^{(m)} + c_3 z_i^{(m)} \right), \\
&= c_0 \left(\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \right) + c_1 \left(\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} x_i^{(m)} \right) + c_2 \left(\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} y_i^{(m)} \right) + \\
&\quad + c_3 \left(\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} z_i^{(m)} \right), \\
&= c_0 \left(\sum_{i=0}^{N_{en}-1} v_{\tilde{K}}^{\boldsymbol{\xi}_i} \right) + c_1 x + c_2 y + c_3 z,
\end{aligned}$$

by using both the definition of reference map and nodal interpolant. □

The convergence condition 2.12 has to be verified case by case, but it is quite simple to satisfy.

2.3 One-dimensional problems

As a simple introduction to the framework, one-dimensional problems are presented. In 2.4 this presentation will be generalized to two-dimensional problems.

2.3.1 Analysis with lowest-order elements

We start from the weak formulation for a one-dimensional elliptic problem, which is the same written in a more general form in (1.13):

$$a(\upsilon, \varphi) = l(\varphi), \ \forall \varphi \in H^1(\Omega) \tag{2.8}$$

with

$$a(\upsilon, \varphi) = \int_\Omega (a_1 \upsilon' \cdot \varphi' + a_0 \upsilon \varphi)\, dx$$

$$l(\varphi) = \int_\Omega (f\varphi - a_1 \gamma' \cdot \varphi' - a_0 \gamma \varphi)\, dx + \left[a_1 (\upsilon + \gamma)' \varphi \right]_a^b$$

equipped with the boundary conditions

$$\upsilon = 0, \ \forall x \in \Gamma_D$$

$$(\upsilon + \gamma)' = g_N, \ \forall x \in \Gamma_N$$

where $u = \upsilon + \gamma$ and $\gamma = g_D$ in Γ_D, a sequence of steps is needed to get an approximation.

Mesh In one dimension, the domain Ω can be as the open interval (a, b), which can then be partitioned in M_n intervals

$$a = x_{0,n} < \ldots < x_{M_n,n} = b.$$

These intervals form the so called mesh of the domain Ω in one dimension

$$\mathcal{M}_n = \{K_{1,n}, \ldots, K_{M_n,n}\}$$

where

$$K_{i,n} = (x_{i-1,n}, x_{i,n})$$

is the i^{th} nodal finite element (described in general form in 2.2) and the x_i's are called *grid vertices.* It is also possible to define the mesh diameter $h(n)$ as

$$h(n) = \max_{1 \le i \le M_n} (x_{i-1,n} - x_{i,n}).$$

Application of the Galerkin method The Galerkin subspace $V_n \subset V$ consists (in FEM) of piecewise-polynomial functions of degree $p_i \geq 1$ defined on the finite elements

$$V_n = \{v \in V : v \in P^{p_i}(K_{i,n}) \, \forall i = 1, \ldots, M_n\}. \tag{2.9}$$

A good choice of basis functions consists of functions with small support for which as many as possible of them are disjoint. This way the term $a(v_i, v_j)$ is zero each time v_i and v_j are disjoint, generating a sparse stiffness matrix, which is far simpler and more efficient to handle by computers.

The most used basis for the space V_n is the one formed by $M_n - 1$ functions of degree $p_m = 1$, $\forall m$ defined by (suppose from now on that every x_i and K_i is referred to the current mesh so that $x_i = x_{i,n}$ and $K_i = K_{i,n}$):

$$v_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & x \in \overline{K}_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i}, & x \in \overline{K}_{i+1} \\ 0, & \{x \in (a,b) \,|\, x \notin \overline{K}_i \vee x \notin \overline{K}_{i+1}\} \end{cases}, \; i = 1, \ldots, M_n - 1. \tag{2.10}$$

The shape functions

$$v_0 = \frac{x_0 - x}{x_1 - x_0}, \; x_0 \leq x \leq x_1,$$

$$v_{M_n} = \frac{x - x_{M_n}}{x_{M_n} - x_{M_n - 1}}, \; x_{M_n - 1} \leq x \leq x_{M_n},$$

are related to the boundary nodes. All these functions are called *hat functions* or *roof functions* and, when used in FEM, are named *basis* or *shape* or *vertex functions.* In this case the space V_n is a piecewise linear finite element space. Given this space, the function v_n is written

$$v_n = \sum_{i \in G \setminus G_D} \bar{v}_i v_i,$$

where

$$G = \{0, \ldots, M_n\},$$

$$G_D = \{i | i = 0, \ldots, M_n,\ x_i \in \Gamma_D\}.$$

The system of algebraic equations is written with the following matrices

$$\boldsymbol{S}_n = \left[\int_\Omega \left(a_1 v_i' \cdot v_j' + a_0 v_i v_j\right) dx\right]_{i,j}, \tag{2.11}$$

$$\bar{\boldsymbol{\Upsilon}}_n = \left[\bar{v}_i\right]_i,$$

$$\boldsymbol{F}_n = \left[\int_\Omega \left(f v_i - a_1 \gamma' \cdot v_i' - a_0 \gamma v_i\right) dx + \left[a_1 \left(\upsilon + \gamma\right)' v_i\right]_a^b\right]_i, \tag{2.12}$$

where $i, j \in G \backslash G_D$. A possible choice for γ is

$$\gamma(x) = \sum_{i \in G_D} u(x_i)\, v_i(x).$$

The i^{th} nodal finite element is therefore defined by the triple $\left(K_i, P^1(K_i), \mathcal{L}\right)$ and is called *lowest-order element*.

Example 2.23. Consider the problem of solving

$$\begin{cases} u'' = 0,\ x \in \Omega = (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases}.$$

It is possible to find a classical solution $u \in C^2(\Omega) \cap C\left(\bar{\Omega}\right)$ integrating twice both members:

$$\iint_\Omega u'' dx dx = \iint_\Omega 0 dx dx \Rightarrow u(x) = c_1 x + c_2$$

and then imposing the boundary conditions:

$$\begin{cases} u(x) = c_1 x + c_2,\ x \in \Omega = (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases} \Rightarrow u(x) = x. \tag{2.13}$$

The problem is equipped with nonhomogeneous Dirichlet boundary conditions, so it is necessary to define a Dirichlet lift γ and the new function υ so that $u = \gamma + \upsilon$. A possible choice for γ is

$$\gamma = 0 \cdot v_0 + 1 \cdot v_{M_n},$$

where v_0 is a roof function centered on the left boundary of Ω and v_{M_n} a roof function centered on the right boundary of Ω. By substitution in Equation (2.14) the new problem reads ($u = u(x)$ and $v = v(x)$):

$$\begin{cases} (\gamma + \upsilon)'' = 0,\ x \in \Omega = (0,1) \\ \upsilon(0) = 0 \\ \upsilon(1) = 0 \end{cases}.$$

It is possible furthermore to derive a weak formulation of the problem following the four steps of sub-subsection 1.4.1. Multiply by $\varphi \in C_0^\infty(\Omega)$:

$$(\gamma + \upsilon)'' \varphi = 0.$$

Then integrate over Ω:

$$\int_{\Omega} (\gamma + \upsilon)'' \varphi dx = 0$$

and reduce the maximum degree of the derivatives using the technique of integration by parts[2] :

$$\int_{\Omega} (\gamma + \upsilon)'' \varphi dx = \left[\varphi (\gamma + \upsilon)'\right]_{\partial\Omega} - \int_{\Omega} \varphi' (\gamma + \upsilon)' dx.$$

Making use of the last equation and since $\varphi \in C_0^{\infty}(\Omega)$ vanishes on the boundary, the equation can be simplified to

$$\int_0^1 \varphi' v' dx = -\int_0^1 \varphi' \gamma'. \tag{2.14}$$

It is necessary now to use the Galerkin method to obtain a discrete problem from (2.14). Approximating the problem in a subspace V_n with $\dim(V_n) = M_n - 1$ we get the matrices:

$$\boldsymbol{S}_n = \left[\int_0^1 v_i' v_j' dx\right]_{i,j=1}^{M_n-1},$$

$$\bar{\boldsymbol{\Upsilon}}_n = [\bar{v}_i]_{i=1}^{M_n-1},$$

$$\boldsymbol{F}_n = \left[-\int_0^1 v_i' \gamma' dx\right]_{i=1}^{M_n-1},$$

where the v_i's are the roof functions defined in (2.10). If a partition of the domain Ω with $M_n - 1$ internal nodes is built, then the linear system $\boldsymbol{S}_n \cdot \bar{\boldsymbol{\Upsilon}}_n = \boldsymbol{F}_n$ can be solved for $\bar{\boldsymbol{\Upsilon}}_n$. The approximate solution is then

$$u_n(x) = \sum_{i=1}^{M_n-1} \bar{v}_i v_i(x) + \gamma.$$

In this particular case, $u_n \equiv u$.

Consider the problem

$$\begin{cases} \dfrac{1}{50} \cdot u'' = x, \ x \in \Omega = (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases}.$$

The same procedure of Example 2.23 can be followed to obtain the exact solution and the discrete formulation. The exact solution is

$$u(x) = \frac{x}{3}\left(25x^2 - 22\right).$$

[2] If $f(x) = f$ and $g(x) = g$ are two continuously differentiable functions then:

$$\int_a^b fg' dx = [fg]_a^b - \int_a^b f' g dx.$$

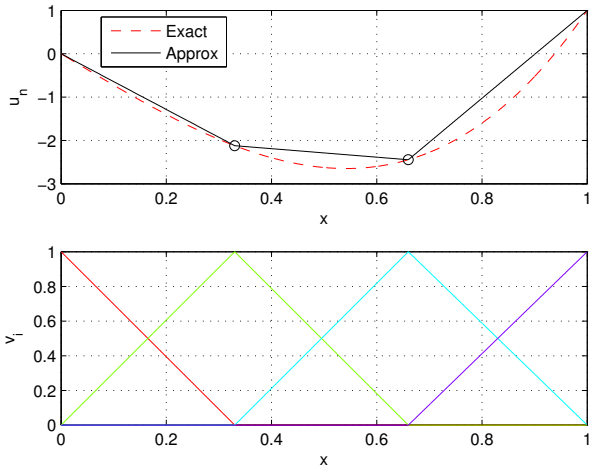

Figure 2.1: Representation of the exact solution (red curve) and of the approximate (black curve). The plot at the bottom represents the basis functions v_i used in FEM.

If a mesh with three elements is considered, the linear system is $\boldsymbol{S}_n \cdot \bar{\boldsymbol{\Upsilon}}_n = \boldsymbol{F}_n$ where

$$\boldsymbol{S}_n = \left[\int_0^1 -\frac{1}{50} v_i' v_j' dx\right]_{i,j=1}^2,$$

$$\bar{\boldsymbol{\Upsilon}}_n = [\bar{v}_i]_{i=1}^2,$$

$$\boldsymbol{F}_n = \left[\int_0^1 \left(x v_i + \frac{1}{50} v_i' \gamma'\right) dx\right]_{i=1}^2.$$

Figure 2.1 represents the exact solution compared with the approximate.

2.3.2 Transformation of the model to the reference domain

Once the stiffness matrix and the force vector are found, like in Equation (2.12) and (2.11), it is sufficient to compute the integrals (typically approximation methods can be used) and solve the linear system. However, that of Subsection 2.3.1 is a very simple case, in which we were able to define all the shape functions and it is easy to compute the integrals. There are cases in which it is not simple to define the shape functions and it is more efficient to change the formulation using a reference map like stated in Subsection 2.2.1. We can try to do this for this simple case as well.

So, our task is to define the reference map $x_{K_m} : K_a \to K_m$ where $K_a = (\xi_0, \xi_1) = (-1, 1)$ is our reference domain. Currently, our vertex functions on the reference domain can be simply found (they are piecewise-polynomials, and in this section we employ a degree 1), and are

$$v_{K_a}^{\xi_0}(\xi) = -\frac{(\xi - 1)}{2}, \; v_{K_a}^{\xi_1}(\xi) = \frac{\xi + 1}{2}.$$

Once the shape functions are defined on the reference domain, the isoparametric concept can be invoked, and we can express the reference map with

$$x_{K_m}(\xi) = \sum_{i=0}^{1} v_{K_a}^{\xi_i}(\xi)\, x_{i,m}.$$

The result of this summation can be written as

$$x_{K_m}(\xi) = c_1^{(m)} + c_2^{(m)}\xi,$$

where

$$c_1^{(m)} = \frac{x_{m-1} + x_m}{2},\ c_2^{(m)} = J_{K_m} = \frac{x_m - x_{m-1}}{2},$$

supposing $K_m = (x_{m-1}, x_m)$ as said before.

2.3.2.1 Finite element space

With the reference map, it is possible to redefine the space V_n defined in (2.9), (the notation with the n is left for the notation with $h = h(n)$ to remark the relation to the chosen mesh)

$$V_{h,p} = \{v \in V : v|_{K_m} \in P^{p_m}(K_m)\ \forall m = 1, 2, \ldots, M_{h,p}\}$$

with

$$V_{h,p} = \{v \in V : (v|_{K_m} \circ x_{K_m})(\xi) \in P^{p_m}(K_a)\ \forall m = 1, 2, \ldots, M_{h,p}\},$$

i.e. $V_{h,p}$ is the set containing all the functions in V defined in each domain K_m which, once transformed to the reference domain, is a polynomial of degree p_m on K_a. As stated above, it is no more needed to define shape functions on each domain. It is only necessary to have one reference map for each domain ad a set of shape functions on the reference domain.

2.3.2.2 Transformation of the equation to the reference domain

Consider again, for instance, the model of the problem (1.13): the next step is to transform the weak form to the reference domain, so as to take advantage of the reference map. The bilinear and the linear forms can be calculated using the two following integrals:

$$a(\upsilon, \varphi) = \int_\Omega (a_1 \upsilon' \cdot \varphi' + a_0 \upsilon\varphi)\, dx,$$

$$l(\varphi) = \int_\Omega (f\varphi - a_1\gamma' \cdot \varphi' - a_0\gamma\varphi)\, dx + \left[a_1(\upsilon + \gamma)'\varphi\right]_a^b.$$

The first thing to do is to divide the integrals into a summation over the elements as $\Omega = \bigcup_{m=1}^{M_{h,p}} K_m$:

$$a(\upsilon, \varphi) = \sum_{m=1}^{M_{h,p}} \int_{K_m} (a_1 \upsilon' \cdot \varphi' + a_0 \upsilon\varphi)\, dx,$$

$$l(\varphi) = \sum_{m=1}^{M_{h,p}} \int_{K_m} (f\varphi - a_1\gamma' \cdot \varphi' - a_0\gamma\varphi)\, dx + \left[a_1 (\upsilon + \gamma)' \varphi\right]_a^b.$$

Then, using the result (2.4) of the Galerkin method, the problem can be rewritten to: find the function $\upsilon_{h,p} \in V_{h,p}(\Omega)$ so that

$$\upsilon_{h,p} = \sum_{i \in G \backslash G_D} \bar{v}_i v_i,$$

where

$$\operatorname{span}\left(\{v_j\}_{j=0}^{M_{h,p}}\right) = V_{h,p}(\Omega),$$

and

$$\sum_{i \in G \backslash G_D} \bar{v}_i \sum_{m=1}^{M_{h,p}} \int_{K_m} \left(a_1 v_i' v_j' + a_0 v_i v_j\right) dx =$$

$$= \sum_{m=1}^{M_{h,p}} \int_{K_m} \left(f v_j - a_1 v_i' v_j' - a_0 \gamma v_j\right) + \left[a_1 (\upsilon + \gamma)' v_j\right]_a^b,\ j \in G \backslash G_D.$$

Transformation of functions to the reference domain Functions are transformed to the reference domain by simply composing them with the reference map:

$$\tilde{a}_l^{(m)}(\xi) = (a_l \circ x_{K_m})(\xi) = a_l(x_{K_m}(\xi)),\ l = 0, 1$$

$$\tilde{v}_l^{(m)}(\xi) = (v_l \circ x_{K_m})(\xi) = v_l(x_{K_m}(\xi)),\ l = i, j$$

$$\tilde{f}(\xi) = (f \circ x_{K_m})(\xi) = f(x_{K_m}(\xi)),$$

$$\tilde{\gamma}(\xi) = (\gamma \circ x_{K_m})(\xi) = \gamma(x_{K_m}(\xi)).$$

Transformation of derivatives to the reference domain The transformation of the derivative can be found using the chain rule

$$\left(\tilde{v}_l^{(m)}(\xi)\right)' = (v_l \circ x_{K_m})'(\xi) = v_l'(x)\,|_{x = x_{K_m}(\xi)} J_{K_m}(\xi),\ l = i, j,$$

$$\left(\tilde{\gamma}^{(m)}(\xi)\right)' = (\gamma \circ x_{K_m})'(\xi) = \gamma'(x)\,|_{x = x_{K_m}(\xi)} J_{K_m}(\xi).$$

Transformation of integrals to the reference domain The transformation of the integrals is simple applying the Substitution theorem. The application of the Substitution theorem allows the transformation of the bilinear form

$$a(v_i, v_j) = \sum_{m=1}^{M_{h,p}} \int_{K_m} \left(a_1 v_i' v_j' + a_0 v_i v_j\right) dx$$

to the reference domain. Let f be

$$f(x) = a_1(x)\, v_i'(x)\, v_j'(x) + a_0(x)\, v_i(x)\, v_j(x).$$

We can write

$$a(v_i, v_j) = \sum_{m=1}^{M_{h,p}} \int_{x_{K_m}(\tilde{K})} f(x)\, dx = \sum_{m=1}^{M_{h,p}} \int_{\tilde{K}} f(x_{K_m}(\xi))\, J_{x_{K_m}} d\xi,$$

where f can be expressed as

$$f(x_{K_m}(\xi)) = \frac{1}{J^2_{x_{K_m}}} \tilde{a}_1^{(m)} \left(\tilde{v}_i^{(m)}\right)' (\tilde{v}_j)' + \tilde{a}_0^{(m)} \tilde{v}_i^{(m)} \tilde{v}_j^{(m)}.$$

Following the same process we can write

$$l(v_j) = \sum_{m=1}^{M_{h,p}} \int_{\tilde{K}} g(x_{K_m}(\xi))\, J_{x_{K_m}} d\xi + \left[a_1 (\upsilon + \gamma)' \varphi\right]_a^b$$

where g can be expressed as

$$g(x_{K_m}(\xi)) = \tilde{f}^{(m)} \tilde{v}_j^{(m)} - \tilde{a}_0^{(m)} \tilde{\gamma}^{(m)} \tilde{v}_j^{(m)} + \frac{1}{J^2_{\tilde{K}_m}} \tilde{a}_1^{(m)} \left(\tilde{\gamma}^{(m)}\right)' \left(\tilde{v}_j^{(m)}\right)'.$$

2.3.3 Exactness at the nodes

It is important to remark that it's possible to prove that the values found for the DOFs are exact.

Theorem 2.24. *$u(x_i) = u_h(x_i)$, $i = 0, \ldots, M_{h,p}$.*

Proof. The proof of this theorem can be found in [17]. □

2.3.4 Higher-order elements

It is possible to consider piecewise-polynomials (i.e. polynomials defined on the elements) of degree higher than 1. The elevation of the degree is a type of refinement named *p*-refinement as it should lead to approximations closer to the exact solution, leaving the mesh unchanged.

2.3.4.1 Lagrange nodal shape functions

The construction of shape functions is an important task in FEM as a wrong basis could make the solution of the linear system difficult. One approach is that which employs the Lagrange interpolation. The task is to find a basis $\mathcal{P} = \{l_{Lag,1}, l_{Lag,1}, \ldots, l_{Lag,p_m+1}\}$ in the space $P^{p_m}\left(\tilde{K}\right)$, defining the shape functions on the reference domain. The shape functions are associated to an equal number of nodal points defined on the reference domain (suppose it is $\tilde{K} = (-1, 1)$) using the Lagrange interpolation conditions $l_{Lag,j}(y_k) = \delta_{jk}$. The shape functions can be defined by the equation

$$l_{Lag,i}(\xi) = \prod_{1 \le j \le p_m+1, j \ne i} \frac{(\xi - y_j)}{(y_i - y_j)},\ i = 1, 2, \ldots, p_m + 1.$$

Algorithm 2.1 computes the Lagrange interpolating polynomials for the given points to be interpolated.

Algorithm 2.1 Algorithm to compute the Lagrange polynomial interpolating the points provided.

```
% lagrangePoly determines the Lagrange interpolating polynomial
% interpolating the points provided.
% Input:
%   X: x coordinates of the points to be interpolated;
%   Y: y coordinates of the points to be interpolated;
%   XX: optional argument to get the values of the polynomial in the
%       values contained in the vector XX.
% Output:
%   P: coefficients of the polynomial in highest order first;
%   R: x coordinates of the N-1 extrema of the resulting polynomial;
%   S: y coordinates of the extrema.
function [P, R, S] = lagrangePoly(X, Y, XX)
% Make sure that X and Y are row vectors
if size(X,1) > 1;  X = X'; end
if size(Y,1) > 1;  Y = Y'; end
if size(X,1) > 1 || size(Y,1) > 1 || size(X,2) ~= size(Y,2)
  error('both inputs must be equal-length vectors')
end
N = length(X);
pvals = zeros(N,N);
% Calculate the polynomial weights for each order
for i = 1:N
  % the polynomial whose roots are all the values of X except this one
  pp = poly(X( (1:N) ~= i));
  % scale so its value is exactly 1 at this X point (and zero
  % at others, of course)
  pvals(i,:) = pp ./ polyval(pp, X(i));
end
% Each row gives the polynomial that is 1 at the corresponding X
% point and zero everywhere else, so weighting each row by the
% desired row and summing (in this case the polycoeffs) gives
% the final polynomial
P = Y*pvals;
if nargin==3
  % output is YY corresponding to input XX
  YY = polyval(P,XX);
  % assign to output
  P = YY;
end
if nargout > 1
  % Extra return arguments are values where dy/dx is zero
  % Solve for x s.t. dy/dx is zero i.e. roots of derivative polynomial
  % derivative of polynomial P scales each power by its power, downshifts
  R = roots( ((N-1):-1:1) .* P(1:(N-1)) );
  if nargout > 2
    % Calculate the actual values at the points of zero derivative.
    S = polyval(P,R);
  end
end
```

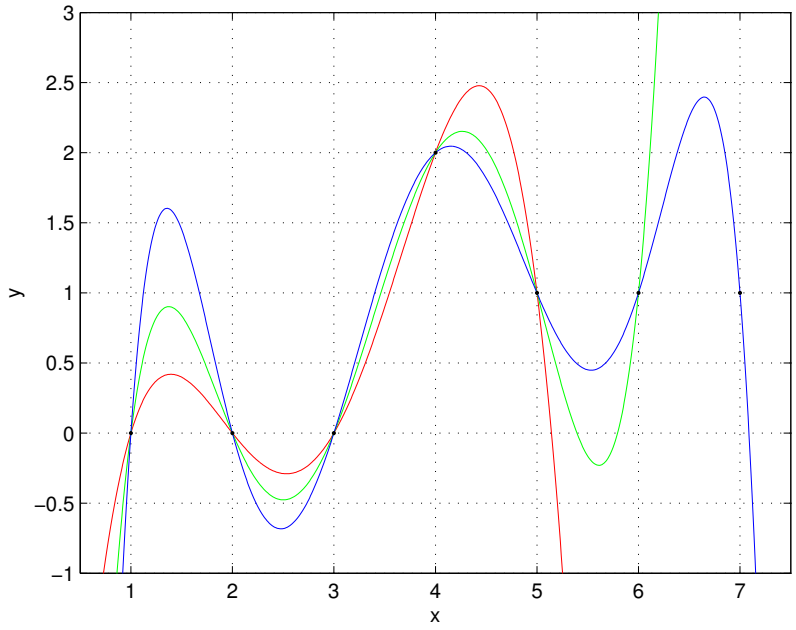

Figure 2.2: Example of the effects of the Runge's phenomenon. Langrange interpolating polynomials interpolating 4, 5 and 6 points.

2.3.4.2 Runge's phenomenon

Analyzing the function

$$f(x) = \frac{1}{1+25x^2},$$

Runge found that, interpolating at equidistant points between -1 and 1 with polynomials $P_n(x)$ of increasing degree n, the interpolation oscillates towards the end of the interval. It can be proven, in fact, that

$$\lim_{n\to\infty} \left(\max_{-1\leq x\leq 1} |f(x) - P_n(x)| \right) = +\infty.$$

An example of the effects caused by the Runge's phenomenon can be seen in Figure 2.2. However, the *Weierstrass approximation theorem* turns out to be useful in this regard:

Theorem 2.25. *Suppose f is a continuous complex-valued function defined on the real interval $[a,b]$. For every $\epsilon > 0$, there exists a polynomial function P over $\mathbb{C}$ such that for all x in $[a,b]$, we have $|f(x) - P(x)| < \epsilon$, or equivalently, the supremum norm $\|f(x) - P(x)\| < \epsilon$. If f is real-valued, the polynomial function can be taken over $\mathbb{R}$.*

This means there is some sequence of polynomials for which the error goes to zero. So, using equidistant nodal points is a possible solution, but there are better choices: *Chebyshev nodal points* and *Gauss-Lobatto nodal points.*

2.3.4.3 Chebyshev and Gauss-Lobatto nodal points

Assuming we want to define nodal shape functions of degree $p_m > 1$ in the domain $\tilde{K} = (-1, 1)$, the Chebyshev points are defined by the equation

$$y_j = \cos\left(\frac{\pi j}{p}\right),\ j = 0, 1, \ldots, p_m,$$

whereas the Gauss-Lobatto nodal points are the roots of the function

$$\left(1 - x^2\right) L'_{p_m}(x).$$

The function $L_{p_m}(x)$ is the *Legendre polynomial* of degree p_m which can be calculated using the *Rodrigues' formula*

$$L_{p_m}(x) = \frac{1}{2^{p_m} \cdot p_m!} \cdot \frac{d^{p_m}\left(\left(x^2 - 1\right)^{p_m}\right)}{dx^{p_m}}. \tag{2.15}$$

The Legendre polynomials are the solutions to the Legendre differential equation

$$\frac{d}{dx}\left(\left(1 - x^2\right) \cdot \frac{d}{dx} L_k(x)\right) + n(n+1) L_k(x) = 0.$$

This ODE is commonly encountered in physics and other technical fields.

Shape functions associated with grid vertices are called *vertex functions*, shape functions associated with nodal points are called *bubble functions*.

Nodal basis of the space $V_{h,p}$ Given the nodal shape functions $l_{Lag,i}(\xi)$, $i = 1, 2, \ldots, p_m + 1$, it is possible to redefine the choice for the Galerkin subspace $V_{h,p}$. It is common to assume that all nodal finite elements have the same degree, but this is not mandatory, so suppose it is p_m for the nodal finite element K_m. A basis of the space can be made up by $M_{h,p} - 1$ *vertex functions* ($M_{h,p}$ is the number of nodal finite elements)

$$v_i = \begin{cases} \left(l_{Lag,p_m+1} \circ x_{K_i}^{-1}\right)(x), & x \in K_i \\ \left(l_{Lag,0} \circ x_{K_{i+1}}^{-1}\right)(x), & x \in K_{i+1} \end{cases}$$

and by the $\sum_{m=1}^{M_{h,p}} (p_m - 1)$ *bubble functions*

$$\left(l_{Lag,2} \circ x_{K_m}^{-1}\right)(x),\ \left(l_{Lag,3} \circ x_{K_m}^{-1}\right)(x),\ \ldots,\ \left(l_{Lag,p_m} \circ x_{K_m}^{-1}\right)(x).$$

2.3.4.4 Lobatto hierarchic shape functions

An alternative way of building a basis is to use a hierarchical approach: an initial basis is defined, and bases of higher dimension are built adding functions to the previous basis defined. An example of good hierarchic shape functions for elliptic problems in one dimension are the Lobatto polynomials

$$l_{Lob,0}(\xi) = \frac{1 - \xi}{2},\ l_{Lob,1}(\xi) = \frac{1 + \xi}{2},$$

$$l_{Lob,k} = \int_{-1}^{\xi} L_{k-1}(\psi)\, d\psi,\ 2 \le k,$$

where L_{k-1} is the Legendre polynomial which can be calculated using (2.15).

Nodal basis of the space $V_{h,p}$ As already done for the Lagrange shape functions, once the functions are defined, a new space $V_{h,p}$ can be constructed. The nodal *vertex functions* are defined by the equation

$$v_i(x) = \begin{cases} \left(l_{Lob,1} \circ x_{K_i}^{-1}\right)(x), & x \in K_i \\ \left(l_{Lob,0} \circ x_{K_{i+1}}^{-1}\right)(x), & x \in K_{i+1} \end{cases},$$

which indicates they are linear functions, the nodal *bubble functions* are defined by

$$\left(l_{Lob,2} \circ x_{K_m}^{-1}\right)(x), \left(l_{Lob,3} \circ x_{K_m}^{-1}\right)(x), \ldots, \left(l_{Lob,p_m} \circ x_{K_m}^{-1}\right)(x).$$

2.4 Two-dimensional problems

In most cases problems are not one-dimensional but two- or three-dimensional. We have, therefore, to define new elements to partition the geometry (simple intervals can be used only in one-dimensional problems).

2.4.1 $\mathcal{K}_q^1$-elements

The $\mathcal{K}_q^1$-element is a well known nodal finite element used to create meshes on two-dimensional domains.

Definition 2.26. A $\mathcal{K}_q^1$*-element* is a nodal finite element defined by the triple $\left(K, Q^1(K), \mathcal{L}\right)$ where:

- K is a quadrilateral domain;
- $Q^1(K)$ is the space of the polynomials of first degree defined on the domain K;
- $\mathcal{L}$ is the set of degrees of freedom.

We need to define in particular the $\mathcal{K}_q^1$-element on the reference domain as required by the concept explained in Subsection 2.2.1.

Definition 2.27. A $\mathcal{K}_q^1$-element defined on the reference domain is the element $\mathcal{K}_q^{1,r}$ defined by the triple $\left(K_q, Q^1(K_q), \mathcal{L}_q\right)$ where:

- K_q is the domain $(-1,1)^2$;
- $Q^1(K_q) = \text{span}(\{1, \xi, \eta, \xi\eta\})$;
- $\mathcal{L}_q = \{L_i\}_{i=1}^4$ where $g \in Q^1(K_q)$ and $L_i : Q^1(K_q) \to \mathbb{R}$ are

$$L_i(g) = g(\boldsymbol{\xi}_i),\ i = 1, 2, \ldots, 4,$$

$$\boldsymbol{\xi}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ \boldsymbol{\xi}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ \boldsymbol{\xi}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix},\ \boldsymbol{\xi}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Lemma 2.28. *The nodal finite element* $\left(K_q, Q^1(K_q), \mathcal{L}_q\right)$ *is unisolvent, and the nodal basis of the space* $Q^1(K_q)$ *consists of the vertex functions*

$$v_{K_q}^{\boldsymbol{\xi}_1}(\boldsymbol{\xi}) = \frac{(1-\xi)(1-\eta)}{4},\ v_{K_q}^{\boldsymbol{\xi}_2}(\boldsymbol{\xi}) = \frac{(1+\xi)(1-\eta)}{4},$$

$$v_{K_q}^{\boldsymbol{\xi}_3}(\boldsymbol{\xi}) = \frac{(1-\xi)(1+\eta)}{4},\ v_{K_q}^{\boldsymbol{\xi}_4}(\boldsymbol{\xi}) = \frac{(1+\xi)(1+\eta)}{4}.$$

Proof. The unisolvency can be proved by building the generalized Vandermonde matrix $\boldsymbol{L}$, which is found to be nonsingular. The nodal basis can be obtained by computing the inverse of $\boldsymbol{L}$. □

We want to create the change of coordinates

$$\boldsymbol{\xi} = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \underset{\boldsymbol{x}_{K_m}^{-1}}{\overset{\boldsymbol{x}_{K_m}}{\rightleftarrows}} \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

by way of a reference map $\boldsymbol{x}_{K_m} : K_q \to K_m$. It's time to invoke the isoparametric concept defined in Sub-subsection 2.2.1, adapted to the two-dimensional problem. This means we would like to define the reference map through

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}) = \sum_{i=1}^{4} v_{K_q}^{\boldsymbol{\xi}_i} \boldsymbol{x}_i^{(m)}. \tag{2.16}$$

Proposition 2.29. *The reference map of Equation (2.16) satisfies*

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}_i) = \boldsymbol{x}_i^{(m)},\ i = 1, 2, \ldots, 4$$

and

$$\boldsymbol{x}_{K_m}(e_i) = s_i,\ i = 1, 2, \ldots, 4$$

where e_i *is the edge of the element in the reference domain and* s_i *is the edge of the element.*

Proposition 2.30. *The nodal finite element* $\left(K_m, Q^1(K_m), \mathcal{L}_m\right)$ *is unisolvent, and the shape functions*

$$v_{K_m}^{\boldsymbol{x}_i}(\boldsymbol{x}) = \left(v_{K_q}^{\boldsymbol{\xi}_i} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}),\ i = 1, 2, \ldots, 4,$$

constitute a unique nodal basis of the space $Q^1(K_m)$.

We want to determine if this element guarantees the convergence of the Galerkin method. For this, we need to guarantee smoothness of the basis functions, continuity on the boundary and completeness (see 2.2.2).

As said in Subsection 2.2.3, smoothness are almost always assured, unless the element is too distorted. The degenerated triangle was an example case, but even quadrilaterals with interior angles of more than 180° can make this condition to be unsatisfied. This is caused by the fact that the inverse functions $\xi(\boldsymbol{x}_{K_m})$ and $\eta(\boldsymbol{x}_{K_m})$ of the reference map could be not well defined. Away from these cases of high degeneration, $\mathcal{K}_q^1$-elements guarantees smoothness of the shape functions.

We have to analyze the shape function to assure continuity on the boundary. Let's consider, for instance, the shape function defined on the node $\boldsymbol{\xi}_1$ on the edge with $\eta = 1$

$$v_{K_q}^{\boldsymbol{\xi}_1}(\boldsymbol{\xi}) = \frac{1-\xi}{2}.$$

This is a one-dimensional linear shape functions already defined, and the same holds for the other edges. After these considerations of the shape of the function on the edges it is possible to guess that the shape function is an hyperbolic paraboloid, which guarantees continuity on the boundaries.

Proving the completeness is very simple: according to 2.2.3 it is sufficient to prove the sum to 1:

$$\sum_{i=1}^{4} v_{K_q}^{\boldsymbol{\xi}_i} = \frac{(1-\xi)(1-\eta)}{4} + \frac{(1+\xi)(1-\eta)}{4} + \\ + \frac{(1-\xi)(1+\eta)}{4} + \frac{(1+\xi)(1+\eta)}{4} = 1.$$

2.4.2 $\mathcal{K}_t^1$-elements

Another type of well known element in the two-dimensional space is the $\mathcal{K}_t^1$-element.

Definition 2.31. A $\mathcal{K}_t^1$-element is a nodal finite element defined by the triple $\left(K, P^1(K), \mathcal{L}\right)$ where:

- K is a triangular domain;
- $P^1(K)$ is the space of the polynomials of first degree defined on the domain K;
- $\mathcal{L}$ is the set of degrees of freedom.

We need to define in particular the $\mathcal{K}_t^1$-element on the reference domain. One of the possibilities in the definition of a triangular reference domain is the triangle derived from the degeneration of the domain K_q, where the nodes $\boldsymbol{\xi}_3$ and $\boldsymbol{\xi}_4$ are coalesced into one node $\boldsymbol{\xi}_3 = [0,1]^T$. In this case, the resulting shape functions are

$$v_{K_t}^{\boldsymbol{\xi}_1}(\boldsymbol{\xi}) = \frac{(1-\xi)(1-\eta)}{4},\ v_{K_t}^{\boldsymbol{\xi}_2}(\boldsymbol{\xi}) = \frac{(1+\xi)(1+\eta)}{4},\ v_{K_t}^{\boldsymbol{\xi}_3}(\boldsymbol{\xi}) = \frac{1+\eta}{2}.$$

Although the determinant of the Jacobian matrix is zero in $\boldsymbol{\xi}_3$, the derivatives exist with respect to ξ and η. This is sufficient to have smoothness. In the same way we did in 2.4.1, it is possible to show both continuity and completeness. This means such a reference element assures the convergence of the Galerkin method. Anyway, it has to be underlined that the determinant of the Jacobian matrix is not constant. This means that, when integrating on this element, the Jacobian has to be integrated numerically, and is rising the degree of the integrand, which requires more integration points in Gauss quadrature (see Section 2.7).

Another possibility for a reference element is the following:

Definition 2.32. A $\mathcal{K}_t^1$-element defined on the reference domain is the $\mathcal{K}_t^1$-element $\mathcal{K}_t^{1,r}$ defined by the triple $\left(K_t, P^1(K_t), \mathcal{L}_t\right)$ where:

- $K_t = \left\{ [\xi, \eta]^T : \eta < \xi, \eta > -1, \xi > -1 \right\}$;
- $P^1(K_t) = \text{span}(\{1, \xi, \eta\})$;
- $\mathcal{L}_t = \{L_i\}_{i=1}^3$ where $g \in P^1(K_t)$ and $L_i : P^1(K_t) \to \mathbb{R}$ are

$$L_i(g) = g(\boldsymbol{\xi}_i),\ i = 1, 2, 3,$$

$$\boldsymbol{\xi}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ \boldsymbol{\xi}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ \boldsymbol{\xi}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Lemma 2.33. *The nodal finite element* $\left(K_t, P^1(K_t), \mathcal{L}_t\right)$ *is unisolvent, and the nodal basis of the space* $P^1(K_t)$ *consists of the vertex functions*

$$v_{K_t}^{\boldsymbol{\xi}_1}(\boldsymbol{\xi}) = -\frac{\xi + \eta}{2},\ v_{K_t}^{\boldsymbol{\xi}_2}(\boldsymbol{\xi}) = \frac{1 + \xi}{2},\ v_{K_t}^{\boldsymbol{\xi}_3}(\boldsymbol{\xi}) = \frac{1 + \eta}{2}.$$

Proof. See proof of Lemma 2.28. □

As we did in the case of the $\mathcal{K}_q^{1,r}$-element, we have to define the reference map. This is simple to do; invoking the isoparametric concept we get

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}) = \sum_{i=1}^{3} v_{K_t}^{\boldsymbol{\xi}_i}(\boldsymbol{\xi})\, \boldsymbol{x}_i^{(m)}, \tag{2.17}$$

and again the following proposition holds:

Proposition 2.34. *The reference map of Equation (2.17) satisfies*

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}_i) = \boldsymbol{x}_i^{(m)},\ i = 1, 2, 3$$

and

$$\boldsymbol{x}_{K_m}(e_i) = s_i,\ i = 1, 2, 3$$

where e_i *is the edge of the element in the reference domain and* s_i *is the edge of the element with domain* K_t.

Proposition 2.35. *The nodal finite element* $\left(K_m, P^1(K_m), \mathcal{L}_m\right)$ *is unisolvent, and the shape functions*

$$v_{K_m}^{\boldsymbol{x}_i^{(m)}}(\boldsymbol{x}) = \left(v_{K_t}^{\boldsymbol{\xi}_i} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}),\ i = 1, 2, 3$$

constitute a unique nodal basis of the space $P^1(K_m)$.

Again, it is interesting to prove that the element $\mathcal{K}_t$ guarantees the convergence. The continuity across the boundaries can be verified the same way it was done in Subsection 2.4.1. The condition of completeness is automatic as this is an isoparametric element. What's important to note is that the determinant of the Jacobian matrix is now constant

$$J_{\boldsymbol{x}_{K_m}} = \begin{vmatrix} \dfrac{x_{1,2} - x_{1,1}}{2} & \dfrac{x_{1,3} - x_{1,1}}{2} \\ \dfrac{x_{2,2} - x_{1,1}}{2} & \dfrac{x_{2,3} - x_{2,1}}{2} \end{vmatrix}.$$

where it is assumed $\boldsymbol{x}_i = [x_{1,i}, x_{2,i}]$. This is a better choice as, this way, the determinant of the Jacobian matrix doesn't have to be integrated numerically.

2.4.3 Analysis with lowest-order elements

Problems in two dimensions need some more steps than one-dimensional problems. The reference problem expressed in the equation (1.10) is again translated to its weak formulation in (1.13): find a function $v \in H^1(\Omega)$ such that

$$\iint_\Omega (a_1 \nabla v \cdot \nabla \varphi + a_0 v \varphi)\, d\boldsymbol{z} =$$

$$\iint_\Omega (f\varphi - a_1 \nabla \gamma \cdot \nabla \varphi - a_0 \gamma \varphi)\, d\boldsymbol{z} + \int_{\Gamma_N} (a_1 g_N \varphi)\, d\boldsymbol{S},\ \forall \varphi \in H^1(\Omega).$$

What is needed is a process which goes through the approximations needed to generate the linear system of equations $\boldsymbol{S} \cdot \tilde{\boldsymbol{\Upsilon}} = \boldsymbol{F}$. Unfortunately, in the two-dimensional case, some approximations are needed to reach the discrete problem: these approximations are called *variational crimes*.

Approximation of the domain A 2D domain Ω needs, in general, to be approximated with a domain Ω_h. If Ω is a polygonal domain or if the boundary $\partial\Omega$ is piecewise-polynomial, then the approximation can be done exactly. In case $\Omega_h \nsubseteq \Omega$, then a variational crime is committed since the functions could be evaluated where they are not defined.

Mesh The domain Ω_h has to be covered with M_h elements

$$\mathcal{M}_{h,p} = \{K_i, i = 1, \ldots, M_{h,p}\}.$$

$\mathcal{M}_{h,p}$ is the mesh defined on the domain.

Approximation of the boundaries In the first step it was necessary to modify the domain Ω creating another domain $\Omega_{h,p}$. This means the boundary $\partial\Omega$ can be different from the new boundary $\partial\Omega_{h,p}$. A new variational crime arises in case $\partial\Omega \neq \partial\Omega_{h,p}$ as the functions g_D and g_N defined respectively in (1.7) and (1.9) could be undefined on $\partial\Omega_{h,p}$. It is therefore necessary to redefine g_D and g_N on the boundaries of Ω_h, choosing a new suitable partition $\partial\Omega_{h,p} = \Gamma_{D,h,p} \cup \Gamma_{N,h,p}$. The new boundary conditions can be now written:

$$u(\boldsymbol{z}) = g_{D,h,p}(\boldsymbol{z}),\ \forall \boldsymbol{z} \in \Gamma_{D,h,p},$$

$$\frac{\partial u(\boldsymbol{z})}{\partial \boldsymbol{\nu}} = g_{N,h,p}(\boldsymbol{z}),\ \forall \boldsymbol{z} \in \Gamma_{N,h,p}.$$

The Dirichlet lift needs to be approximated as well with a new function $\gamma_{h,p} \in H^1(\Omega_{h,p})$.

Approximation of the Hilbert spaces The space V on which the problem is formulated is defined on the domain Ω. This means it is necessary to redefine the space with another one,

$$V_{h,p} = \{v \in C(\Omega_{h,p}) : v(\boldsymbol{z}) = 0,\ \forall \boldsymbol{z} \in \Gamma_{D.h,p},$$

$$v|_{K_i} \in P^p(K_i) \text{ if } K_i \text{ is a triangle},$$

$$v|_{K_i} \in Q^p(K_i) \text{ if } K_i \text{ is a quarilateral}\}. \tag{2.18}$$

defined on the approximated domain $\Omega_{h,p}$. Unfortunately, it is not possible to guarantee $V_{h,p} \subset V$ as the Galerkin method requires, so that it is possible that another variational crime is committed. The same happens to the trial solutions space:

$$\begin{aligned} U_{h,p} = \{ & v \in C(\Omega_{h,p}) : v(\boldsymbol{z}) = g_{D,h,p},\ \forall \boldsymbol{z} \in \Gamma_{D.h,p}, \\ & v|_{K_i} \in P^p(K_i) \text{ if } K_i \text{ is a triangle}, \\ & v|_{K_i} \in Q^p(K_i) \text{ if } K_i \text{ is a quarilateral} \}. \end{aligned} \tag{2.19}$$

Approximate weak formulation It is necessary to approximate the weak formulation according to the modifications done so far to the model of equation (1.12): It's now required to find the function $v_{h,p} \in V_{h,p}$ and $\gamma_{h,p} \in U_{h,p}$ so that

$$\iint_{\Omega_{h,p}} (a_1 \nabla v_{h,p} \cdot \nabla v + a_0 v_{h,p} v)\, d\boldsymbol{z} =$$

$$\iint_{\Omega_{h,p}} (fv - a_1 \nabla \gamma_{h,p} \cdot \nabla v - a_0 \gamma_{h,p} v)\, d\boldsymbol{z} + \int_{\Gamma_{N,h,p}} a_1 g_{N,h,p} v d\boldsymbol{S},\ \forall v \in V_{h,p}.$$

Application of the Galerkin model The Galerkin method remains almost unchanged: it is possible to express the unknown function $v_{h,p}$ as a linear combination of the basis functions of the space $V_{h,p}$ not related to nodes with Dirichlet boundary conditions. We define

$$G = \{ i | v_i \in \text{a nodal base of } V_{h,p} \},$$

$$G_D = \{ i | v_i \in \text{a nodal base of } V_{h,p},\ \boldsymbol{x}_i \in \Gamma_{D,h,p} \}.$$

The unknown $v_{h,p}$ is then

$$v_{h,p} = \sum_{i \in G \backslash G_D} \bar{v}_i v_i,$$

and the Dirichlet lift can be written as well as

$$\gamma_{h,p} = \sum_{i \in G_D} g_{D,h,p}(\boldsymbol{x}_i)\, v_i,$$

assuming $v_1, \ldots, v_{\dim(V_{h,p})}$ is a nodal basis of the space $V_{h,p}$ with $\dim(V_{h,p}) = N_{h,p}$. The new problem is now expressed by the system

$$\sum_{i \in G \backslash G_D} \bar{v}_i \iint_{\Omega_{h,p}} (a_1 \nabla v_i \cdot \nabla v_j + a_0 v_i v_j)\, d\boldsymbol{z} =$$

$$\iint_{\Omega_{h,p}} (f v_j - a_1 \nabla \gamma_{h,p} \cdot \nabla v_j - a_0 \gamma_{h,p} v_j)\, d\boldsymbol{z} + \int_{\Gamma_{N,h,p}} a_1 g_{N,h,p} v_j d\boldsymbol{S} \tag{2.20}$$

for all $j \in G \backslash G_D$. We can notice as well that we can substitute the choice made of Dirichlet lift

$$\sum_{i \in G \backslash G_D} \bar{v}_i \iint_{\Omega_{h,p}} (a_1 \nabla v_i \cdot \nabla v_j + a_0 v_i v_j)\, d\boldsymbol{z} =$$

$$\iint_{\Omega_{h,p}} f v_j d\boldsymbol{z} - \sum_{i \in G_D} \iint_{\Omega_{h,p}} a_1 g_{D,h,p}(\boldsymbol{x}_i) \nabla v_i \cdot \nabla v_j d\boldsymbol{z} +$$
$$- \sum_{i \in G_D} \iint_{\Omega_{h,p}} a_0 g_{D,h,p}(\boldsymbol{x}_i) v_i v_j d\boldsymbol{z} + \int_{\Gamma_{N,h,p}} a_1 g_{N,h,p} v_j d\boldsymbol{S}.$$

for all $j \in G \backslash G_D$.

2.4.4 Transformation of the model to the reference domain

Like done in Subsection 2.3.2, we would like to use the reference map to work with the model in the reference domain. For both the $\mathcal{K}_q^1$-elements and the $\mathcal{K}_t^1$-elements we've already defined the reference maps. Let's assume an hybrid mesh with $M_{h,p} = M_q + M_t$ nodes, where M_q is the number of $\mathcal{K}_q^1$-elements and M_t is the number of $\mathcal{K}_t^1$-elements, a set $\{\boldsymbol{x}_i,\ i = 1, \ldots, N_{h,p}\}$ of grid vertices that don't belong to $\Gamma_{D,h,p}$ (*unconstrained grid vertices*).

Proposition 2.36. *The dimension of the space $V_{h,p}$ is $N_{h,p}$ which is also the number of unconstrained grid vertices.*

2.4.4.1 Basis of the space $V_{h,p}$

We already defined the space we're working with in Equation (2.19), but we need a basis for this space. This is a little more complicated than in the one-dimensional case: first of all we need to define a *vertex patch* $S(i)$, which will be the support of the shape function. It is defined as the union of all the mesh elements which have $\boldsymbol{x}_i$ among their nodes:

$$S(i) = \bigcup_{k \in N(i)} \bar{K}_m,\ N(i) = \{m : K_m \in \mathcal{M}_{h,p}, \boldsymbol{x}_i \text{ is vertex of } K_m\}. \tag{2.21}$$

Employing the same concept of the one-dimensional case, we define the shape functions to be

$$v_{K_m}^{\boldsymbol{x}_i}(\boldsymbol{x}) = \begin{cases} \left(v_{K_q}^{\boldsymbol{\xi}_r} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_m \text{ and } K_m \text{ is a quadrilateral} \\ \left(v_{K_t}^{\boldsymbol{\xi}_r} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_m \text{ and } K_m \text{ is a triangle} \\ 0, & \boldsymbol{x} \in \Omega_h \setminus S(i) \end{cases}. \tag{2.22}$$

It can be shown that these vertex functions form a basis for the space $V_{h,p}$.

2.4.4.2 Transformation of the equation to the reference domain

Just like we did in Sub-subsection 2.3.2.2, the first step is to split the integral of Equation (2.20) as a summation over the elements K_m:

$$\sum_{i \in G \backslash G_D} \bar{v}_i \sum_{m=1}^{M_{h,p}} \iint_{K_m} (a_1 \nabla v_i \cdot \nabla v_j + a_0 v_i v_j) d\boldsymbol{z} =$$

$$\sum_{m=1}^{M_{h,p}} \iint_{K_m} (f v_j - a_1 \nabla \gamma_{h,p} \cdot \nabla v_j - a_0 \gamma_{h,p} v_j) d\boldsymbol{z} + \sum_{m=1}^{M_{h,p}} \int_{\Gamma_{N,h,p} \cap \overline{K_m}} a_1 g_{N,h,p} v_j d\boldsymbol{S}, \tag{2.23}$$

$j \in G \backslash G_D$.

Transformation of functions to the reference domain The transformation of the model requires first of all to transfer the function to the reference domain. This is done again, like in Sub-subsection 2.3.2.2, composing functions with the $\circ$ operator with the reference map $\boldsymbol{x}_{K_m}(\boldsymbol{\xi}) = (x_{K_m,1}(\boldsymbol{\xi}), x_{K_m,2}(\boldsymbol{\xi}))$. For a function $\psi(\xi,\eta) = \psi(\boldsymbol{\xi})$, this results in:

$$\psi^{(m)}(\boldsymbol{\xi}) = (\psi \circ \boldsymbol{x}_{K_m})(\boldsymbol{\xi}) = \psi(x_{K_m,1}(\boldsymbol{\xi}), x_{K_m,2}(\boldsymbol{\xi})).$$

Transformation of derivatives to the reference domain Again, like in Sub-subsection 2.3.2.2, the chain rule is used:

$$\frac{\partial\tilde{\psi}^{(m)}}{\partial\xi}(\boldsymbol{\xi}) = \frac{\partial\psi}{\partial x}|_{\boldsymbol{x}=\boldsymbol{x}_{K_m}(\boldsymbol{\xi})}\frac{\partial x_{K_m,1}}{\partial\xi}(\boldsymbol{\xi}) + \frac{\partial\psi}{\partial y}|_{\boldsymbol{x}=\boldsymbol{x}_{K_m}(\boldsymbol{\xi})}\frac{\partial x_{K_m,2}}{\partial\xi}(\boldsymbol{\xi})$$

$$\frac{\partial\tilde{\psi}^{(m)}}{\partial\eta}(\boldsymbol{\xi}) = \frac{\partial\psi}{\partial x}|_{\boldsymbol{x}=\boldsymbol{x}_{K_m}(\boldsymbol{\xi})}\frac{\partial x_{K_m,1}}{\partial\eta}(\boldsymbol{\xi}) + \frac{\partial\psi}{\partial y}|_{\boldsymbol{x}=\boldsymbol{x}_{K_m}(\boldsymbol{\xi})}\frac{\partial x_{K_m,2}}{\partial\eta}(\boldsymbol{\xi}).$$

This result can be written in matrix form so that the Jacobian matrix (defined in Definition B.3) can be recognized:

$$\begin{bmatrix} \frac{\partial\tilde{\psi}^{(m)}}{\partial\xi} \\ \frac{\partial\tilde{\psi}^{(m)}}{\partial\eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_{K_m,1}}{\partial\xi} & \frac{\partial x_{K_m,2}}{\partial\xi} \\ \frac{\partial x_{K_m,1}}{\partial\eta} & \frac{\partial x_{K_m,2}}{\partial\eta} \end{bmatrix} \begin{bmatrix} \frac{\partial\psi}{\partial x} \\ \frac{\partial\psi}{\partial y} \end{bmatrix} = \left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^T \begin{bmatrix} \frac{\partial\psi}{\partial x} \\ \frac{\partial\psi}{\partial y} \end{bmatrix}. \tag{2.24}$$

$\left({}^{D\boldsymbol{x}_{K_m}}/_{D\boldsymbol{\xi}}\right)$ is the Jacobian matrix of the reference map $\boldsymbol{x}_{K_m}$ and is the correspondent of the Jacobian found in the one-dimensional case. So, Equation (2.24) can be written in more compact form using the gradient

$$\nabla\tilde{\psi}^{(m)}(\boldsymbol{\xi}) = \left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^T \nabla\psi(\boldsymbol{x}).$$

If the Jacobian is nonsingular, $\nabla\psi$ can be found using

$$\nabla\psi(\boldsymbol{x}) = \left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^{-T} \nabla\tilde{\psi}^{(m)}(\boldsymbol{\xi})$$

where

$$\left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^{-T} = \left(\left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^{-1}\right)^T = \left(\left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^T\right)^{-1}.$$

Transformation of integrals to the reference domain Using the substitution theorem it is possible to change the integration domain using the results above and the fact that $\boldsymbol{x}_{K_m}\left(\tilde{K}\right) = K_m$ where $\tilde{K}$ is either K_t or K_q. The left member of Equation (2.23) can be transformed to

$$\sum_{i=1}^{N_{h,p}} \bar{v}_i \sum_{m=1}^{M_{h,p}} \iint_{\boldsymbol{x}_{K_m}(\tilde{K})} (a_1 \nabla v_i \cdot \nabla v_j + a_0 v_i v_j)\, d\boldsymbol{z} =$$

$$\sum_{i=1}^{N_{h,p}} \bar{v}_i \sum_{m=1}^{M_{h,p}} \iint_{\tilde{K}} J_{\boldsymbol{x}_{K_m}} \left(\tilde{a}_1^{(m)} \left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^{-T} \nabla\tilde{v}_i^{(m)}\right) \left(\left(\frac{D\boldsymbol{x}_{K_m}}{D\boldsymbol{\xi}}\right)^{-T} \nabla\tilde{v}_j^{(m)}\right) d\boldsymbol{\xi}$$

$$+\sum_{i=1}^{N_{h,p}} \bar{v}_i \sum_{m=1}^{M_{h,p}} \iint_{\tilde{K}} J_{\boldsymbol{x}_{K_m}} \tilde{a}_0^{(m)} \tilde{v}_i^{(m)} \tilde{v}_j^{(m)} d\boldsymbol{\xi}.$$

where $|^{D\boldsymbol{x}_{K_m}}/_{D\boldsymbol{\xi}}| = J_{\boldsymbol{x}_{K_m}}$.

2.4.5 Higher-order elements

2.4.5.1 Lagrange-Gauss-Lobatto $\mathcal{K}_q^{p,r}$-elements

$\mathcal{K}_q^{p,r}$-elements are the generalization of $\mathcal{K}_q^1$-elements using Lagrange shape functions. Two- and three-dimensional Lagrange shape functions can be obtained using the product of one-dimensional Lagrange functions. The choice of the points can be taken accordingly to be the product of Gauss-Lobatto points in $\bar{K}_a$ being $K_q = K_a \times K_a$. In the case of the quadrilateral reference domain, each axis direction has a different order of approximation, that's why these elements are indicated with $\mathcal{K}_q^{p,r}$. The points can be divided into three groups: *vertex nodes*, *edge nodes* and *bubble nodes*. Vertex nodes corresponds to the four vertices of the quadrilateral:

$$\boldsymbol{\xi}^{\boldsymbol{\xi}_1} = \boldsymbol{\xi}_1 = \left[x_1^{(p)}, x_1^{(r)}\right]^T = [-1,-1]^T,\ \boldsymbol{\xi}^{\boldsymbol{\xi}_2} = \boldsymbol{\xi}_2 = \left[x_{p+1}^{(p)}, x_1^{(r)}\right]^T = [1,-1]^T,$$

$$\boldsymbol{\xi}^{\boldsymbol{\xi}_3} = \boldsymbol{\xi}_3 = \left[x_1^{(p)}, x_{r+1}^{(r)}\right]^T = [-1,1]^T,\ \boldsymbol{\xi}^{\boldsymbol{\xi}_4} = \boldsymbol{\xi}_4 = \left[x_{p+1}^{(p)}, x_{r+1}^{(r)}\right]^T = [1,1]^T,$$

where the $x_i^{(p)}$'s are the Gauss-Lobatto points of order p and the $x_i^{(r)}$'s are the Gauss-Lobatto points of order r, so $x_i^{(p)} \in \bar{K}_a$ and $x_i^{(r)} \in \bar{K}_a$. There are $r-1$ edge nodes which lie on the edges of the quadrilateral

$$\begin{aligned}
e_1 &= \{\boldsymbol{\xi}_1 + t(\boldsymbol{\xi}_3 - \boldsymbol{\xi}_1) \,|t \in [0,1]\}, \\
e_2 &= \{\boldsymbol{\xi}_2 + t(\boldsymbol{\xi}_4 - \boldsymbol{\xi}_2) \,|t \in [0,1]\}, \\
e_3 &= \{\boldsymbol{\xi}_1 + t(\boldsymbol{\xi}_2 - \boldsymbol{\xi}_1) \,|t \in [0,1]\}, \\
e_4 &= \{\boldsymbol{\xi}_3 + t(\boldsymbol{\xi}_4 - \boldsymbol{\xi}_3) \,|t \in [0,1]\},
\end{aligned}$$

and can be defined as

$$\begin{aligned}
\boldsymbol{\xi}_1^{e_1} &= \left[x_1^{(p)}, x_2^{(r)}\right]^T = \left[-1, x_2^{(r)}\right]^T, \\
\boldsymbol{\xi}_2^{e_1} &= \left[x_1^{(p)}, x_3^{(r)}\right]^T = \left[-1, x_3^{(r)}\right]^T, \\
&\vdots \\
\boldsymbol{\xi}_{r-1}^{e_1} &= \left[x_1^{(p)}, x_r^{(r)}\right]^T = \left[-1, x_r^{(r)}\right]^T,
\end{aligned}$$

and correspondingly for the other nodes. The bubble nodes instead, are defined as

$$\boldsymbol{\xi}_{i,j}^b = \left[x_{i+1}^{(p)}, x_{j+1}^{(r)}\right]^T, i = 1,2,\ldots,p-1, j = 1,2,\ldots,r-1.$$

Our task is to determine a nodal basis of the space

$$Q^{p,r}(K_q) = \left\{\xi^k \eta^l : 1 \le k \le p, 1 \le l \le r, -1 \le \xi, \eta \le 1\right\}.$$

In the same way it was done above for the points, we can divide the shape functions in three groups: the *vertex functions*, the *edge functions* and the *bubble functions*. Each of them is built using the one-dimensional Lagrange functions $l^{(p)}_{Lag,i}, i = 1,2,\ldots,p+1$ and $l^{(r)}_{Lag,j}, j = 1,2,\ldots,r+1$ where p and r are the orders. Each of them are forced to satisfy, as already done in the one-dimensional case, the delta property

$$l^{(p)}_{Lag,k}\left(x^{(p)}_l\right) = \delta_{kl}.$$

The four vertex functions are

$$v^{\boldsymbol{\xi}_1}_{K_q}(\boldsymbol{\xi}) = l^{(p)}_{Lag,1}(\xi)\cdot l^{(r)}_{Lag,1}(\eta),\ v^{\boldsymbol{\xi}_2}_{K_q}(\boldsymbol{\xi}) = l^{(p)}_{Lag,p+1}(\xi)\cdot l^{(r)}_{Lag,1}(\eta),$$

$$v^{\boldsymbol{\xi}_3}_{K_q}(\boldsymbol{\xi}) = l^{(p)}_{Lag,1}(\xi)\cdot l^{(r)}_{Lag,r+1}(\eta),\ v^{\boldsymbol{\xi}_4}_{K_q}(\boldsymbol{\xi}) = l^{(p)}_{Lag,p+1}(\xi)\cdot l^{(r)}_{Lag,r+1}(\eta). \quad (2.25)$$

The edge functions are

$$v^{e_1}_{K_q,1}(\boldsymbol{\xi}) = l^{(p)}_{Lag,1}(\xi)\cdot l^{(r)}_{Lag,2}(\eta) \quad (2.26)$$

$$v^{e_1}_{K_q,2}(\boldsymbol{\xi}) = l^{(p)}_{Lag,1}(\xi)\cdot l^{(r)}_{Lag,3}(\eta) \quad (2.27)$$

$$\vdots \quad (2.28)$$

$$v^{e_1}_{K_q.r-1}(\boldsymbol{\xi}) = l^{(p)}_{Lag,1}(\xi)\cdot l^{(r)}_{Lag,r}(\eta). \quad (2.29)$$

The bubble functions are

$$v^{b}_{K_q,i,j}(\boldsymbol{\xi}) = l^{(p)}_{Lag,i+1}(\xi)\cdot l^{(r)}_{Lag,j+1}(\eta), i = 1,2,\ldots,p-1, j = 1,2,\ldots,r-1. \quad (2.30)$$

Proposition 2.37. *The Lagrange shape functions (2.30), (2.29) and (2.25) form a basis of the space $Q^{p,r}(K_q)$.*

Proposition 2.38. *The Lagrange shape functions (2.30), (2.29) and (2.25) satisfy the delta property and are therefore a nodal basis of the space $Q^{p,r}(K_q)$.*

2.4.5.2 Lagrange-Fekete P^p-elements

Again, we would like to generalize the $\mathcal{K}^1_t$-elements with p^{th}-order elements using Lagrange shape functions. We can do this following the same procedure used in the previous sub-subsection. First of all we have to define suitable and efficient points for the triangular domain. A good choice in this sense are the *Fekete points*. In its original version, the problem consists in determining the position of N points on a compact subset $K \subset \mathbb{R}^2$ that maximize the product of their mutual Euclidean distances. The N_P-tuples $\left(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{N_P}\right)$ that satisfies this property are called N^{th}_P order Fekete points in K. An alternative definition is

Definition 2.39. Let a bounded convex domain $K \subset \mathbb{R}^d$ be equipped with a polynomial space $P(K)$ of dimension N_P. Given an arbitrary basis $\{\vartheta_i\}_{i=1}^{N_P}$ of $P(K)$, the *Fekete points* $\{\boldsymbol{\xi}_i\}_{i=1}^{N_P} \subset K$ are a point set that maximizes the determinant

$$\left|\boldsymbol{L}\left(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{N_P}\right)\right| = \max_{\{\boldsymbol{\xi}_1,\boldsymbol{\xi}_2,\ldots,\boldsymbol{\xi}_{N_P}\}\subset\overline{K}} \left|\boldsymbol{L}\left(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{N_P}\right)\right|,$$

where $\boldsymbol{L}$ is the generalized Vandermonde matrix for the Lagrange degrees of freedom $L_i(g) = g(\boldsymbol{\xi}_i)$,

$$\boldsymbol{L}\left(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{N_P}\right) = \{L_i(\vartheta_j)\}_{i,j=1}^{N_P} = \{\vartheta_j(\boldsymbol{\xi}_i)\}_{i,j=1}^{N_P}.$$

The advantage of the Fekete points is that they can be defined on any subset of $\mathbb{R}^d$. Unfortunately, no direct formula for computing them is available, so a numerical approximation is needed. An algorithm for the estimation of Fekete points has been recently given in [5].

Again, it is possible to divide the Fekete points in a way such that it is possible to simply define hybrid meshes of quadrilaterals and triangles. We use the already defined domain K_t with the edges

$$e_1 = \{\boldsymbol{\xi}_1 + t(\boldsymbol{\xi}_2 - \boldsymbol{\xi}_1) \,|\, t \in [0,1]\},$$
$$e_2 = \{\boldsymbol{\xi}_2 + t(\boldsymbol{\xi}_3 - \boldsymbol{\xi}_2) \,|\, t \in [0,1]\},$$
$$e_3 = \{\boldsymbol{\xi}_3 + t(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_3) \,|\, t \in [0,1]\}.$$

For the construction we're going to report, the following theorem is fundamental:

Theorem 2.40. *Let $p \geq 1$. The Fekete points have to following properties:*

1. *the Fekete points $\{\boldsymbol{y}_i\}_{i=1}^{N_P}$ are invariant under the choice of the basis $\{\vartheta_i\}_{i=1}^{N_P}$;*
2. *in one-dimensional intervals and Cartesian product geometries the Fekete and Gauss-Lobatto points are the same sets;*
3. *on the edges of triangular domains the Fekete points coincide with the one-dimensional Gauss-Lobatto points.*

Using the p^{th}-order Gauss-Lobatto points $x_i^{(p)} \in \bar{K}_a$ and Theorem 2.40, the three vertex nodes can be defined by

$$\boldsymbol{\xi}^{\boldsymbol{\xi}_1} = \boldsymbol{\xi}_1 = \left[x_1^{(p)}, x_1^{(p)}\right]^T = [-1,-1]^T,$$
$$\boldsymbol{\xi}^{\boldsymbol{\xi}_2} = \boldsymbol{\xi}_2 = \left[x_{p+1}^{(p)}, x_1^{(p)}\right]^T = [1,-1]^T,$$
$$\boldsymbol{\xi}^{\boldsymbol{\xi}_3} = \boldsymbol{\xi}_3 = \left[x_1^{(p)}, x_{p+1}^{(p)}\right]^T = [-1,1]^T.$$

The edge nodes can be defined following the orientation of the edges e_i

$$\begin{aligned}
\boldsymbol{\xi}_1^{e_1} &= \left[x_2^{(p)}, x_1^{(p)}\right]^T = \left[x_2^{(p)}, -1\right]^T, \\
\boldsymbol{\xi}_2^{e_1} &= \left[x_3^{(p)}, x_1^{(p)}\right]^T = \left[x_3^{(p)}, -1\right]^T, \\
&\vdots \\
\boldsymbol{\xi}_p^{e_1} &= \left[x_p^{(p)}, x_1^{(p)}\right]^T = \left[x_p^{(p)}, -1\right]^T.
\end{aligned}$$

The bubble nodes can be sorted in any unique way and are denoted $\boldsymbol{\xi}_i^b, i = 1, 1, \ldots, {}^{(p-1)(p-2)}/_2 - 1$.

The unique Lagrange nodal basis is obtained as stated before, inverting the generalized Vandermonde matrix defined in Definition 2.39.

2.4.5.3 Basis of the space $V_{h,p}$

Like it was done before, we have to build a basis for the space $V_{h,p}$: this is a simple task now that we've defined the shape functions on the reference domain. We just have to use the reference map to move the shape functions to the correct domain. So, let's assume we defined an hybrid mesh $\mathcal{M}_{h,p} = \left\{K_1, K_2, \ldots, K_{M_{h,p}}\right\}$, where $K_m, m = 1, 2, \ldots, M_{h,p}$ is either a triangular or a quadrilateral element. What's important to say here is that it is convenient to assume we use only $\mathcal{K}_t^p$- or $\mathcal{K}_q^p$-elements, as the approximation could not be continuous in case the orders were different. Let M_t and M_q denote the number of $\mathcal{K}_t^p$-elements and of $\mathcal{K}_q^p$-elements respectively.

We continue to separate the definitions of the shape functions in vertex basis functions, edge basis functions and bubble basis functions. Each of them are defined by employing the reference maps already defined in Subsubsections 2.4.1 and 2.4.2 for $\mathcal{K}_q^1$- and $\mathcal{K}_t^1$-elements and the basis functions of the spaces $Q^p(K_q)$ and $P^p(K_t)$ defined in Subsubsections 2.4.5.1 and 2.4.5.2.

Vertex basis functions Vertex basis functions can be defined the same way we did before in Sub-subsection 2.4.4.1. Using the definition of vertex patch of Equation (2.21) we get the basis functions of Equation (2.22)

$$v_{K_m}^{\boldsymbol{x}_i}(\boldsymbol{x}) = \begin{cases} \left(v_{K_q}^{\boldsymbol{\xi}_r} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_m \text{ and } K_m \text{ is a quadrilateral} \\ \left(v_{K_t}^{\boldsymbol{\xi}_r} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_m \text{ and } K_m \text{ is a triangle} \\ 0, & \boldsymbol{x} \in \Omega_h \setminus S(i) \end{cases}.$$

In the contest of Sub-subsection 2.4.4.1 those were the only basis functions, but in the current one, those are only the vertex basis functions. This is simply due to the fact that we're dealing with higher-order elements, in which we defined more points on which other basis functions are to be defined.

Edge basis functions The edge basis functions were not defined in the case of first-order elements. A procedure similar to that used for the vertex basis functions can be used. Assume s_j is an edge for which $\boldsymbol{x}_{K_m}(e_l) = s_j$. We define the *edge element patch*, which is the correspondent of the vertex patch, with the equation

$$S_e(j) = \bigcup_{k \in N_e(j)} \overline{K_k},\ N_e(j) = \{k : K_k \in \mathcal{M}_{h,p}, s_j \text{ is an edge of } K_k\}.$$

According to our definitions of $\mathcal{K}_q^p$-elements and $\mathcal{K}_q^p$-elements, each edge s_j contains exactly $p-1$ nodal points $\boldsymbol{x}_m^{s_j}, m = 1, 2, \ldots, p-1$ for which $\boldsymbol{x}_{K_k}^{-1}\left(\boldsymbol{x}_m^{s_j}\right) = \boldsymbol{\xi}_m^{\boldsymbol{x}_{K_k}^{-1}(s_j)}$. With these elements in hand, we can define the basis edge functions

$$v_{K_k,m}^{s_j} = \begin{cases} \left(v_{K_q,r}^{e_l} \circ \boldsymbol{x}_{K_k}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_k \text{ and } K_k \in S_e(j) \text{ is a quadrilateral} \\ \left(v_{K_t,r}^{e_l} \circ \boldsymbol{x}_{K_k}^{-1}\right)(\boldsymbol{x}), & \boldsymbol{x} \in K_k \text{ and } K_k \in S_e(j) \text{ is a triangle} \\ 0, & \boldsymbol{x} \in \Omega_h \setminus S_e(j) \end{cases},$$

where $v_{K_q,r}^{e_l}(\boldsymbol{\xi})$ and $v_{K_t,r}^{e_l}(\boldsymbol{\xi})$ are edge nodal shape functions defined on the reference domain such that

$$v_{K_q,r}^{e_l}\left(\boldsymbol{x}_{K_k}^{-1}\left(\boldsymbol{x}_m^{s_j}\right)\right) = 1,\ v_{K_t,r}^{e_l}\left(\boldsymbol{x}_{K_k}^{-1}\left(\boldsymbol{x}_m^{s_j}\right)\right) = 1.$$

Bubble basis functions The bubble functions are simpler to be defined. Each $\mathcal{K}_q^p$-element contains $(p-1)^2$ bubble nodes and each P^p-element contains $(p-1)(p-2)/2$. By using the reference map $\boldsymbol{x}_{K_k} : \tilde{K} \to K_k$ it is possible to define

$$\boldsymbol{x}_i^b = \boldsymbol{x}_{K_k}\left(\boldsymbol{\xi}_i^b\right)$$

to be the bubble nodes in the domain K_k using an index only, obtained by the mapping of the bubble points in the reference domain $\tilde{K}$. The bubble functions can be defined as

$$v_{K_k,m}^b(\boldsymbol{x}) = \begin{cases} \left(v_{K_q,r}^b \circ \boldsymbol{x}_{K_k}^{-1}\right)(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in K_k \text{ and } K_k \text{ is a quadrilateral} \\ \left(v_{K_t,r}^b \circ \boldsymbol{x}_{K_k}^{-1}\right)(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in K_k \text{ and } K_k \text{ is a triangle} \\ 0, & \text{if } \boldsymbol{x} \in \Omega_h \setminus K_k \end{cases}.$$

2.5 Three-dimensional problems

There are also elements to be defined for three-dimensional analysis. The simplest element is the brick, but there are also other elements that can be used in many cases when the brick is not adequate.

2.5.1 $\mathcal{K}_B^1$-elements

Let's call $\mathcal{K}_B^1$-element the element used to create meshes on three-dimensional domains (*brick element*). It's definition is:

Definition 2.41. A $\mathcal{K}_B^1$-element is a nodal finite element defined by the triple $\left(K, B^1(K), \mathcal{L}\right)$ where:

- K is an hexahedral domain;
- $B^1(K)$ is the space of the polynomials of first degree defined on the domain K;
- $\mathcal{L}$ is the set of degrees of freedom.

The definition of the $\mathcal{K}_B^1$-element on the reference domain is:

Definition 2.42. A $\mathcal{K}_B^1$-element on the reference domain is the $\mathcal{K}_B^1$-element $\mathcal{K}_B^{1,r}$ defined by the triple $\left(K_B, B^1(K_B), \mathcal{L}_B\right)$ where:

- K_B is the domain $(-1,1)^3$;
- $B^1(K_B) = \text{span}\left(\{1, \xi, \eta, \zeta, \xi\eta, \eta\zeta, \xi\zeta, \xi\eta\zeta\}\right)$;
- $\mathcal{L}_B = \{L_i\}_{i=1}^8$ where $g \in B^1(K_r)$ and $L_i : B^1(K_B) \to \mathbb{R}$ are

$$L_i(g) = g(\boldsymbol{\xi}_i), i = 1, 2, \ldots, 8,$$

$$\boldsymbol{\xi}_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \boldsymbol{\xi}_2 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \boldsymbol{\xi}_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \boldsymbol{\xi}_4 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix},$$

$$\boldsymbol{\xi}_5 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \boldsymbol{\xi}_6 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \boldsymbol{\xi}_7 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \boldsymbol{\xi}_8 = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}.$$

As usual we have to design the nodal basis for the element. We accomplish this task in the proof of Lemma 2.28.

The nodal finite element $\left(K_B, B^1(K_B), \mathcal{L}_B\right)$ is unisolvent, and the nodal basis of the space $B^1(K_B)$ consists of the vertex functions

$$v_{K_B}^{\boldsymbol{\xi}_1}(\boldsymbol{\xi}) = \frac{(1-\xi)(1-\eta)(1-\zeta)}{8},\ v_{K_B}^{\boldsymbol{\xi}_2}(\boldsymbol{\xi}) = \frac{(1+\xi)(1-\eta)(1-\zeta)}{8},$$

$$v_{K_B}^{\boldsymbol{\xi}_3}(\boldsymbol{\xi}) = \frac{(1+\xi)(1+\eta)(1-\zeta)}{8},\ v_{K_B}^{\boldsymbol{\xi}_4}(\boldsymbol{\xi}) = \frac{(1-\xi)(1+\eta)(1-\zeta)}{8},$$

$$v_{K_B}^{\boldsymbol{\xi}_5}(\boldsymbol{\xi}) = \frac{(1-\xi)(1-\eta)(1+\zeta)}{8},\ v_{K_B}^{\boldsymbol{\xi}_6}(\boldsymbol{\xi}) = \frac{(1+\xi)(1-\eta)(1+\zeta)}{8},$$

$$v_{K_B}^{\boldsymbol{\xi}_7}(\boldsymbol{\xi}) = \frac{(1+\xi)(1+\eta)(1+\zeta)}{8},\ v_{K_B}^{\boldsymbol{\xi}_8}(\boldsymbol{\xi}) = \frac{(1-\xi)(1+\eta)(1+\zeta)}{8}.$$

Proof. We have to build the generalized Vandermonde matrix using the basis of the space $B^1(K_B)$ already defined in Definition 2.42

$$\boldsymbol{L} = \begin{bmatrix} 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}.$$

This matrix is invertible and the inverse is

$$\boldsymbol{L}^{-1} = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \end{bmatrix}.$$

This means the element is unisolvent and we can define the nodal basis of this element by looking at the columns of $\boldsymbol{L}^{-1}$. □

Now that we know the definitions of the shape functions defined on the reference domain, it is possible to build the reference map, which is needed to transfer the shape functions on the reference domain to the various elements and to build this way a basis of the space $V_{h,p}$. In the three-dimensional case we want to define a change of coordinates of the type

$$\boldsymbol{\xi} = \begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix} \underset{\boldsymbol{x}_{K_m}^{-1}}{\overset{\boldsymbol{x}_{K_m}}{\rightleftarrows}} \boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

As already done, the isoparametric concept can be invoked, so that the starting point is the definition of the reference map through the use of the shape functions defined on the reference domain:

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}) = \sum_{i=1}^{8} v_{K_B}^{\boldsymbol{\xi}_i} \boldsymbol{x}_i^{(m)}. \tag{2.31}$$

Proposition 2.43. *The reference map of Equation (2.31) satisfies*

$$\boldsymbol{x}_{K_m}(\boldsymbol{\xi}_i) = \boldsymbol{x}_i^{(m)},\ i = 1, 2, \ldots, 8$$

and

$$\boldsymbol{x}_{K_m}(e_i) = s_i,\ i = 1, 2, \ldots, 8$$

where e_i is the edge of the element in the reference domain and s_i is the edge of the element.

Proposition 2.44. *The nodal finite element $\left(K_B, B^1(K_B), \mathcal{L}_B\right)$ is unisolvent, and the shape functions*

$$v_{K_m}^{\boldsymbol{x}_i}(\boldsymbol{x}) = \left(v_{K_m}^{\boldsymbol{\xi}_i} \circ \boldsymbol{x}_{K_m}^{-1}\right)(\boldsymbol{x}), i = 1, 2, \ldots, 8$$

constitute a unique basis of the space $B^1(K_B)$.

The same that has been demonstrated before with $\mathcal{K}_q$ and $\mathcal{K}_t$ can be demonstrated for the three-dimensional $\mathcal{K}_B$ element. The shape functions show continuity across boundaries: this could be seen considering the behavior of the shape function $v_{K_B}^{\boldsymbol{\xi}_1}$ on the face $\zeta = -1$

$$v_{K_B}^{\boldsymbol{\xi}_1}(\xi, \eta, -1) = \frac{(1-\xi)(1-\eta)2}{8} = \frac{(1-\xi)(1-\eta)}{4}.$$

This is an hyperbolic paraboloid and is determined uniquely by the four nodes of the face, hence, the same shape is guaranteed for any other element of this type sharing the same surface. Continuity of the shape function is therefore assured. If the distortion of the element is not too high, the shape functions will be smooth. The completeness of the shape functions is guaranteed by the fact that this element is isoparametric. Therefore, this element assures the convergence of the Galerkin method.

2.6 Higher-order reference maps

We defined isoparametric elements as objects for which a reference map was built using its shape functions. In the subsequent sections anyway, higher-order elements have been introduced, which had new types of shape functions associated with their definition (edge functions and bubble functions). These shape functions can be used to extend the previous definition to use all these new shape functions and not only vertex functions. For quadrilateral elements, for instance:

Definition 2.45. Let $\mathcal{K}_m$ be a quadrilateral element belonging to the mesh $\mathcal{M}_{h,p}$, whose reference element is $\mathcal{K}_q^{1,r}$, local directional orders of approximation $p^{b,1}$, $p^{b,2}$ in the element interior and local polynomial orders p^{e_1}, p^{e_2}, ..., p^{e_4} on the edges. Let $\boldsymbol{x}_{K_m} : \mathcal{K}_q^{1,r} \to K_m$ be of the form

$$\begin{aligned}\boldsymbol{x}_{K_m}(\boldsymbol{\xi}) &= \sum_{i=1}^{4} \boldsymbol{\alpha}_{K_m}^{\boldsymbol{\xi}_i} v_{\tilde{K}_q}^{\boldsymbol{\xi}_i}(\boldsymbol{\xi}) + \\ &+ \sum_{i=1}^{4} \sum_{j=1}^{p^{e_i}-1} \boldsymbol{\alpha}_{K_m,j}^{e_i} v_{\tilde{K}_q,j}^{e_i}(\boldsymbol{\xi}) + \\ &+ \sum_{n_1=1}^{p^{b,1}-1} \sum_{n_2=1}^{p^{b,2}-1} \boldsymbol{\alpha}_{K_m,n_1,n_2}^{b} v_{\tilde{K}_q,n_1,n_2}^{b}(\boldsymbol{\xi}),\end{aligned}$$

where the functions were already defined and are the nodal vertex functions, the nodal edge functions and the nodal bubble functions. If the element interpolation function $u_{h,p}$ can be written using the same functions combined with the DOFs, the element is said to be *isoparametric*. Similar definitions can be given for any other reference element.

2.7 Higher-order numerical quadrature

The numerical evaluation of integrals is fundamental in FEM as both the stiffness matrix and the force vector are determined in general by computing integrals. In some simple cases, these integrals can be evaluated exactly, but this cannot be done in general. The problem of obtaining a precise value is nontrivial and important as, besides obvious effects on the preciseness of the results of the procedure, it could even affect the solvability of the system resulting from the Galerkin method, as the matrix may even become singular.

2.7.1 Quadrature on the reference domain K_a

The general idea of numerical quadrature is to transform someway the integral in a sum. Let $g(y)$ be a function to be integrated over $[a, b]$ with $a < b$

$$\int_a^b g(y)\, dy \approx \sum_{i=0}^{k} A_{k,i} g(y_{k,i}), \tag{2.32}$$

where k is the *order* of the quadrature, the $A_{i,k}$'s are the *quadrature coefficients* and the $y_{i,k}$'s are the *quadrature nodes*. However, all our integrals are evaluated over a specific domain, which is the reference domain. We did this by using an affine reference map (see Subsection 2.2.1). The integral of Equation (2.32) can be transformed to the reference domain K_a as we've already done

$$\int_{-1}^{1} f(\xi)\, d\xi \approx \sum_{i=0}^{k} w_{k,i} f(\xi_{k,i}).$$

We followed the Substitution theorem so $f(\xi) = g(x_{K_m}(\xi))$ and $w_{k,i} = {}^{A_{k,i}}/_{|J_{x_{K_m}}|}$.

A possible way of choosing quadrature nodes and coefficients is that of the Gauss quadrature. The k-point Gauss quadrature reads

$$\int_{-1}^{1} f(\xi)\, d\xi \approx \sum_{i=1}^{k} w_{k,i} f(\xi_{k,i}).$$

This means we have to determine $2k$ parameters: k weights and k nodes. This can be done by solving a system of $2k$ equations which has to be solved for the weights and the nodes: we can use $2k$ functions whose integral is known, like the polynomials $1, \xi, \xi^2, \ldots, \xi^{2k-1}$, to create the nonlinear system of equations

$$\begin{aligned}
\int_{-1}^{1} 1 d\xi &= \sum_{i=1}^{k} w_{k,i}, \\
\int_{-1}^{1} \xi d\xi &= \sum_{i=1}^{k} w_{k,i}\xi_{k,i}, \\
&\vdots \\
\int_{-1}^{1} \xi^{2k-1} d\xi &= \sum_{i=1}^{k} w_{k,i}\xi_{k,i}^{2k-1}.
\end{aligned}$$

Once this system has been solved, both the $\xi_{k,i}$'s and the $w_{k,i}$'s have been computed. As, according to the construction, this method can integrate all the polynomials $1, \xi, \xi^2, \ldots, \xi^{2k-1}$, which forms a basis of the space $P^{2k-1}((-1,1))$, it is simple to see that it can integrate exactly all the polynomials in that space.

Proposition 2.46. *Using the k-point Gaussian quadrature rule it is possible to exactly integrate any polynomial of degree $d \leq 2k-1$.*

Proof. Suppose we want to integrate over $(-1,1)$ the polynomial $p(\xi) = \sum_{j=0}^{d} a_j \xi^j$. So, we want to compute

$$\int_{-1}^{1} \sum_{j=0}^{d} a_j \xi^j d\xi.$$

This can be rewritten using the linearity of the integral

$$a_0 \int_{-1}^{1} \xi^0 d\xi + a_1 \int_{-1}^{1} \xi^1 d\xi + \ldots + a_d \int_{-1}^{1} \xi^d d\xi,$$

and transformed by substitution to

$$a_0 \sum_{i=1}^{k} w_{k,i} + a_1 \sum_{i=1}^{k} w_{k,i}\xi_{k,i} + \ldots + a_d \sum_{i=1}^{k} w_{k,i}\xi_{k,i}^{d}.$$

With simple algebraic manipulations

$$\begin{aligned}\int_{-1}^{1}\sum_{j=0}^{d}a_j\xi^j d\xi &= a_0\sum_{i=1}^{k}w_{k,i}+a_1\sum_{i=1}^{k}w_{k,i}\xi_{k,i}+\ldots+a_d\sum_{i=1}^{k}w_{k,i}\xi_{k,i}^d\\ &= \sum_{i=1}^{k}w_{k,i}a_0+\sum_{i=1}^{k}w_{k,i}a_1\xi_{k,i}+\ldots+\sum_{i=1}^{k}w_{k,i}a_d\xi_{k,i}^d\\ &= \sum_{i=1}^{k}w_{k,j}\left(a_0+a_1\xi_{k,i}+\ldots+a_d\xi_{k,i}^d\right)\\ &= \sum_{i=1}^{k}w_{k,j}\sum_{j=0}^{d}a_j\xi_{k,i}^j\\ &= \sum_{i=1}^{k}w_{k,j}p\left(\xi\right),\end{aligned}$$

we prove the exactness of the computation on the domain $K_a=(-1,1)$. Using one of the designed reference maps and the Substitution theorem it is possible to extend the same to any domain. □

It is moreover possible to prove that the integration points are the roots of the Legendre polynomials $L_k(\xi)$, which can be calculated with Equation (2.15) and that the weights can be calculated with

$$w_{k,i}=\frac{2}{\left(1-\xi_{k,i}^2\right)L_k'\left(\xi\right)^2}.$$

This is an interesting result as the roots of the Legendre polynomials are quite well tabulated.

2.7.2 Quadrature on the reference domain K_q

The Cartesian product can be used to obtain an integration formula for two-dimensional domains like $K_q=K_a\times K_a$. If we integrate on K_a with the formula

$$\int_{K_a}f\left(\xi\right)d\xi\approx\sum_{i=1}^{n_a}w_{n_a,i}f\left(\xi_{n_a,i}\right),$$

we can extend this to integrate exactly all the bivariate polynomials of degree $d_1\leq 2n_a^{(1)}-1$ on the ξ axis and $d_1\leq 2n_a^{(2)}-1$ on the η direction:

$$\begin{aligned}\iint_{K_a^2}g\left(\xi,\eta\right)d\xi d\eta &\approx \int_{-1}^{1}\sum_{i=1}^{n_a^{(1)}}g\left(\xi_{n_a^{(1)},i},\eta\right)w_{n_a^{(1)},i}d\eta\\ &\approx \sum_{j=1}^{n_a^{(2)}}\sum_{i=1}^{n_a^{(1)}}w_{n_a^{(1)},i}w_{n_a^{(2)},j}g\left(\xi_{n_a^{(1)},i},\eta_{n_a^{(2)},j}\right).\end{aligned}$$

Example 2.47. Consider the integral which arises when computing an element of the stiffness matrix of a two-dimensional problem. A possible form of this integral could be $\int_{-1}^{1}\int_{-1}^{1}\xi^2\eta^2 d\xi d\eta$. Exact integration on this yields

$$\int_{-1}^{1}\int_{-1}^{1}\xi^2\eta^2 d\xi d\eta = \int_{-1}^{1}\frac{2}{3}\eta^2 d\eta = \frac{2}{3}\cdot\frac{2}{3} = \frac{4}{9}.$$

Using, instead, a two-point Gauss quadrature

$$\begin{aligned}\int_{-1}^{1}\int_{-1}^{1}\xi^2\eta^2 d\xi d\eta &= \int_{-1}^{1}\left(1\cdot\left(\frac{1}{\sqrt{3}}\right)^2 + 1\cdot\left(\frac{1}{\sqrt{3}}\right)^2\right)\eta^2 d\eta \\ &= \frac{2}{3}\cdot\left(1\cdot\left(\frac{1}{\sqrt{3}}\right)^2 + 1\cdot\left(\frac{1}{\sqrt{3}}\right)^2\right) \\ &= \frac{4}{9}.\end{aligned}$$

2.7.3 Quadrature on the reference domain K_t

The integration on the triangular domain K_t needs a translation to the domain K_q in order to perform the integration using the Gaussian quadrature. The following proposition illustrates how to perform the integration of a function defined on K_t.

Proposition 2.48. *Let K_t be defined on the $\boldsymbol{\xi} = [\xi_1, \xi_2]$-plane and K_q be defined on the $\boldsymbol{\eta} = [\eta_1, \eta_2]$-plane. Let $g(\boldsymbol{\xi})$ be a function defined on the reference domain K_t. Then it is possible to transform the integral over K_t to the integral over K_q:*

$$\iint_{K_t} g(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \iint_{K_q}\frac{1-\eta_2}{2} g\left(-1+\frac{1-\eta_2}{2}(\eta_1+1), \eta_2\right) d\boldsymbol{\eta}.$$

Proof. The proposition can be proved using the mapping

$$\boldsymbol{\xi}(\boldsymbol{\eta}) : \boldsymbol{\eta} \to \boldsymbol{\xi} = \begin{bmatrix} -1+\dfrac{(1-\eta_2)(\eta_1+1)}{2} \\ \eta_2 \end{bmatrix}$$

and the Substitution theorem. □

Once the transformation has been accomplished, it is possible to use what already explained in Subsection 2.7.2.

2.7.4 Quadrature on the reference domain K_B

Once again, the same extension used in Subsection 2.7.2 can be used to integrate over the brick K_B. So

$$\begin{aligned}
\iiint_{K_a^3} g(\xi,\eta,\zeta)\,d\xi d\eta d\zeta &\approx \iint_{K_a^2} \sum_{i=1}^{n_a^{(1)}} g\left(\xi_{n_a^{(1)},i},\eta,\zeta\right) w_{n_a^{(1)},i} d\eta d\zeta \\
&\approx \int_{K_a} \sum_{i=1}^{n_a^{(1)}} \sum_{j=1}^{n_a^{(2)}} g\left(\xi_{n_a^{(1)},i},\eta_{n_a^{(2)},j},\zeta\right) w_{n_a^{(1)},i} w_{n_a^{(2)},j} d\zeta \\
&\approx \sum_{i=1}^{n_a^{(1)}} \sum_{j=1}^{n_a^{(2)}} \sum_{l=1}^{n_a^{(3)}} g\left(\xi_{n_a^{(1)},i},\eta_{n_a^{(2)},j},\zeta_{n_a^{(3)},l}\right) w_{n_a^{(1)},i} w_{n_a^{(2)},j} w_{n_a^{(3)},l}.
\end{aligned}$$

Here again, it is possible to integrate exactly every polynomial of three independent variables up to degrees $2n_a^{(1)}-1$, $2n_a^{(2)}-1$ and $2n_a^{(3)}-1$ in the three variables by choosing $n_a^{(1)}$, $n_a^{(2)}$ and $n_a^{(3)}$ as orders of integration in the three directions.

2.7.5 Choice of the order of numerical integration

When performing numerical integration in finite element analysis, two fundamental problems arise: how many integration points and what kind of numerical integration should we choose? The Gauss numerical integration presented so far has proved to be very efficient, requiring, considering the same precision, less function evaluations than other methods. However, there are cases where other numerical methods of integration could be useful.

Having chosen the Gauss integration method, it has already been explained how to determine both the integration points and weights, but the problem of choosing the order of the integration has been left open. The choice of the order is clearly a trade-off: a high order integration gives a more accurate result, but implies an increase of the cost of the integration. On the other hand, a lower order of numerical integration could lead to a faster integration, but with the penalty of a reduced accuracy. This reduced accuracy could even turn out to be very damaging, as it could lead to a higher number of zeros in the stiffness matrix, making it a singular matrix. A practical way to evaluate the order of numerical integration can be illustrated in the following example.

Example 2.49. Consider the case in which we have to integrate a function $\boldsymbol{F}$ to calculate a the $(i,j)^{\text{th}}$ element of a stiffness matrix $k_{i,j} = \int_{K_m} \boldsymbol{F} d\boldsymbol{x}$. Assuming $\boldsymbol{F} = f\left(x^2, xy, y^2\right)$ is a polynomial and K_m is a quadrilateral, according to the model presented so far, we can transform our integral to the reference domain K_q, getting $k_{i,j} = \int_{K_q} \det\left(J_{K_m}\right) \boldsymbol{F}\left(\boldsymbol{x}_{K_m}(\boldsymbol{\xi})\right) d\boldsymbol{\xi}$. In this case, the determinant of the Jacobian matrix is constant, and therefore it doesn't increase the degree of the integrand. This implies that a two-points Gauss rule is sufficient in each direction to integrate exactly.

As a general rule, given a function $\boldsymbol{F}$ of degree p in each direction, $^{p+1}/_{2}$ or $^{p+2}/_{2}$ evaluations of the integrand $\boldsymbol{F}$ are sufficient to exactly integrate the

function (assuming the determinant of the Jacobian matrix is constant) when p is odd or even respectively.

It should be noted that the Jacobian matrix is not always constant, for instance it is not when the elements are not rectangles or parallelograms. In that case higher orders of numerical integrations are needed.

2.8 Generalization of the Finite Element concepts

In engineering, many possible problems arise when treating PDEs, and the situations reported so far are only the most common. A more general framework can be proposed: we seek an unknown function $\boldsymbol{u}(\boldsymbol{z})$ that satisfies a system of PDEs, which can be wrote with

$$\boldsymbol{A}(\boldsymbol{u}) = [A_1(\boldsymbol{u}), A_2(\boldsymbol{u}), \ldots]^T = \boldsymbol{0},$$

with the boundary conditions

$$\boldsymbol{B}(\boldsymbol{u}) = [B_1(\boldsymbol{u}), B_2(\boldsymbol{u}), \ldots]^T = \boldsymbol{0}.$$

The sought approximated function is

$$\boldsymbol{u} \approx \boldsymbol{u}_h = \sum_{i=1}^{n} \boldsymbol{N}_i \boldsymbol{a}_i = \boldsymbol{N}\boldsymbol{a}.$$

FEM works on the weak form however, so what we have to find is the integral form

$$\int\!\cdots\!\int_{\Omega} \boldsymbol{G}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} + \oint_{\partial\Omega} \boldsymbol{g}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} = \boldsymbol{0},\ \forall j = 1, \ldots, n,$$

where $\boldsymbol{G}_J$ and $\boldsymbol{g}_j$ are functions with the usual structure. It is then possible to write the integrals as a summation over the elements

$$\int\!\cdots\!\int_{\Omega} \boldsymbol{G}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} + \oint_{\partial\Omega} \boldsymbol{g}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} =$$

$$\sum_{m=1}^{M_h} \left(\int\!\cdots\!\int_{K_m} \boldsymbol{G}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} + \oint_{\partial\Omega_m} \boldsymbol{g}_j(\boldsymbol{u}_h)\, d\boldsymbol{z} \right).$$

Two approaches are available: the *method of weighted residuals* and the method of the *variational functionals* for which stationery is sought. A good reference for both is [34].

2.8.1 Method of weighted residuals

The method of the weighted residuals is a procedure similar to that explained in 1.4.1, where the weighting functions $\boldsymbol{\varphi}_i \in V$. To obtain an approximate solution we have to chose an approximation of this space $V_h \subset V$ and of the space of the trial solutions $U_h \subset U$. The choice of the space V_h leads to different methods with different error properties. When $V_h = U_h$, the method is called Galerkin method; when $\boldsymbol{\varphi}_i = \boldsymbol{\delta}_i$, where $\boldsymbol{\delta}_i(\boldsymbol{x}) = \boldsymbol{0}$ for $\boldsymbol{x} \neq \boldsymbol{x}_i$, the method is called *point collocation* and when $\boldsymbol{\varphi}_i = \mathbf{I}$ in Ω_{K_j} and zero elsewhere, the method is named *subdomain collocation*.

2.9 Error estimates and convergence rate

An interesting subject to talk about is the error committed when approximating u with u_n. We define the error of the approximation u_n as $e_n = u - u_n$. The first important thing to note is what's stated in the following lemma.

Lemma 2.50. *Let $u \in V$ be the exact solution of the continuous problem (2.1) and let $u_n \in V_n$ be the solution of the discrete problem (2.3). Then, the error e_n satisfies the equation*

$$a(u - u_n, v) = 0, \ \forall v \in V_n.$$

If the bilinear form is symmetric, the energetic inner product can be introduced as $\langle u, v \rangle_e = a(u, v)$. According to Lemma 2.50 then we get

$$\langle e_n, v \rangle_e = 0, \ \forall v \in V_n.$$

This equation means that the error of the approximation we achieved e_n is orthogonal[3] to every functions of the Galerkin subspace V_n (we say e_n is orthogonal to the subspace V_n). It can be seen as well that u_n is the orthogonal projection[4] of the exact solution $u \in V$ to the Galerkin subspace V_n. As a consequence, u_n is actually the nearest element in V_n to the exact solution $u \in V$ in the energy norm (see A.20). This can be translated to

$$\|u - u_n\|_e = \inf_{v \in V_n} \|u - v\|_e .$$

With these elements in hand, it is possible to formulate the *Céa's lemma*, which is interesting to prove:

Lemma 2.51. *Let V be an Hilbert space, $a(\cdot, \cdot) : V \times V \to \mathbb{R}$ a bilinear bounded V-elliptic form, $l \in V'$, $u \in V$ be the solution to problem (2.1), V_n a subspace of V and $u_n \in V_n$ the solution of the discrete problem (2.3) (Galerkin problem). Let C_a and $\tilde{C}_a$ be the continuity and V-ellipticity constants of the bilinear form $a(\cdot, \cdot)$ (see A.19). Then it is true that*

$$\|u - u_n\|_V \leq \frac{C_a}{\tilde{C}_a} \inf_{v \in V_n} \|u - v\|_V .$$

Proof. We start the proof by considering that the bilinear form $a(u - u_n, u - u_n)$ can be written as the sum of $a(u - u_n, u - v)$ and $a(u - u_n, v - u_n)$, since we can exploit the linearity on the second argument getting

$$\begin{aligned} a(u - u_n, u - v) + a(u - u_n, v - u_n) &= a(u - u_n, u - v + v - u_n) \\ &= a(u - u_n, u - u_n) . \end{aligned}$$

We can state that $v - u_n \in V_n$, as both v and u_n are in V_n (V_n is a linear space and $v - u_n$ is a linear combination which must therefore be in V_n). Thus, $a(u - u_n, v - u_n) = 0$ as a consequence of Lemma 2.50, and we get

$$a(u - u_n, u - v) = a(u - u_n, u - u_n) .$$

[3] Two functions f and g are said to be *orthogonal* if the inner product $\langle f, g \rangle = 0$ whenever $f \neq g$.

[4] If P is an *orthogonal projection*, it projects $v \in V$ onto Pv orthogonally then $\langle v - Pv, u \rangle_e = 0, \ \forall u \in \mathrm{img} P$.

By the V-ellipticity property we have that

$$a\left(u-u_n, u-u_n\right) = \tilde{C}_a \left\|u-u_n\right\|_V^2,$$

and by the boundedness of the bilinear form we have that

$$a\left(u-u_n, u-u_n\right) \leq C_a \left\|u-u_n\right\|_V \left\|u-v\right\|_V, \ \forall v \in V_n.$$

Putting the last two equations together we get

$$\left\|u-u_n\right\|_V \leq \frac{C_a}{\tilde{C}_a} \left\|u-v\right\|_V, \ \forall v \in V_n.$$

□

The importance of the Céa's lemma lies in the fact that it shows the error e_n is independent on the basis chosen. The only thing that matters is the Galerkin subspace chosen. Anyway, one has to be careful when choosing a basis of the subspace as, even if not affecting the error, it affects the performance of the algorithm by conditioning the stiffness matrix.

From what has been said, it is clear that as long as the mesh elements become smaller and smaller, the approximated solution gets nearer and nearer to the exact. In fact, as $n \to +\infty$, $h\left(n\right) \to 0$, which indicates the size of the elements decreases. In the limit, where $h = 0$, the exact solution is determined. However, it is possible to obtain the exact solution after a finite number of subdivisions of the mesh. If the polynomials used in the elements can exactly fit the exact solution, it is possible to get $u_n = u$ after a finite number of subdivision of the mesh, i.e. when $h \neq 0$. Thus, for instance, if the exact solution is of the form of a quadratic polynomial and the shape functions include all the polynomials of that order, the approximation will yield the exact result. Following this fact, we can use the Taylor theorem (see Section B.4) to express the exact solution as a polynomial in the vicinity of a point $\boldsymbol{x}_0$

$$\boldsymbol{u}\left(\boldsymbol{x}\right) = \boldsymbol{u}\left(\boldsymbol{x}_0\right) + \left(\frac{\partial \boldsymbol{u}}{\partial x}\left(\boldsymbol{x}_0\right)\right)\left(x-x_0\right) + \left(\frac{\partial \boldsymbol{u}}{\partial x}\left(\boldsymbol{x}_0\right)\right)\left(y-y_0\right) + \ldots$$

This way, with an element of size h and degree p, a polynomial expansion of degree p can be locally fitted exactly. Since $\boldsymbol{x}-\boldsymbol{x}_0$ is of the order of magnitude of h, the error will be of the order $O\left(h^{p+1}\right)$.

2.10 Adaptive finite element refinement

In Section 2.9 we developed some estimates of the error committed in the approximation of the exact solution. What we need to discuss, is how to reduce this error and how to know the way to reduce it under a specified threshold. We call the process of looking for a solution of lower error *refinement*. When the refinement we're trying to perform is based on the results of previous computations, this refinement is called *adaptive*.

As stated in the section 2.1, the Galerkin method converges when considering subspaces for which (2.2) holds. Hence, assuming we have a space V_n associated

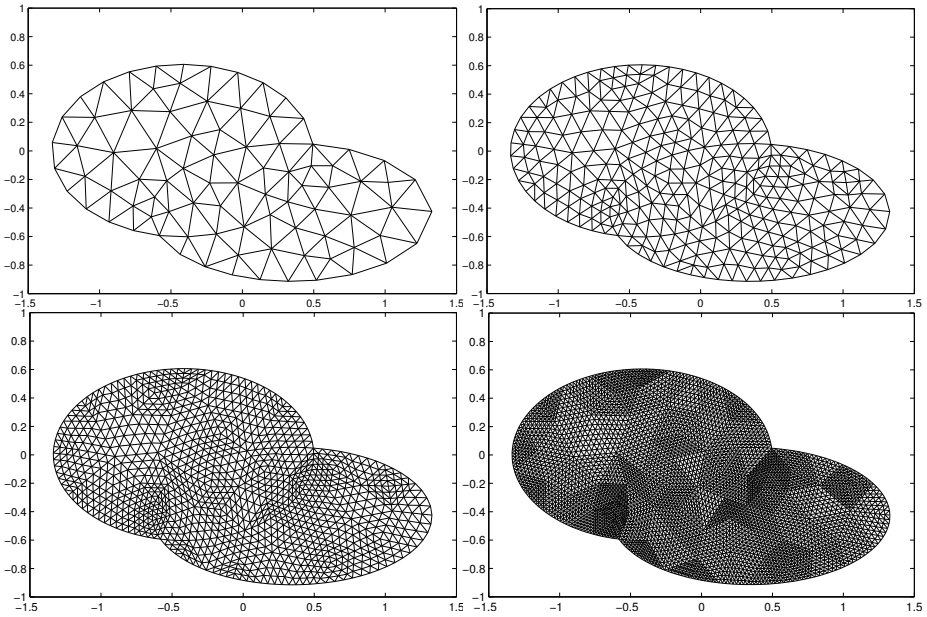

Figure 2.3: h-refinement of a two-dimensional mesh.

with a mesh $\mathcal{M}_n$, reducing the size of the elements we get a new mesh $\mathcal{M}_{n+1}$ (see Figures 2.3 and 2.4) on which the space V_{n+1} satisfies

$$V_n \subset V_{n+1} \subset V.$$

Iterating refinements and calculating approximated solutions on spaces of higher dimension, it is possible to get nearer and nearer to the exact solution (see Figure 2.4). This type of refinement is named *h-refinement*. Unfortunately, as the number of nodes increases, the computational load needed to compute the approximated solution grows exponentially. It is therefore necessary to refine the mesh cleverly.

Reducing the size of the elements is not the only way to accelerate the convergence: it is possible as well to increase the order of the polynomial used in their definition (*p-refinement*).

It is moreover possible to divide the previous categories in subclasses: three typical categories of h-refinement are present and two p-refinement strategies are recognizable. The categories of h-refinement are presented below.

1. The first category of h-refinement is the *element subdivision* (*enrichment*). The refinement in this case is simply implemented subdividing the elements showing too high errors in smaller elements. This method is not very efficient as, during the mesh refinement, some new useless nodes can be created, and the calculation needed to avoid it becomes more involved.

2. Another h-refinement class requires a complete *mesh regeneration* or *remeshing*. In this case, the mesh is completely regenerated, starting from the definition of the domain. This refinement clearly is expensive, particularly in three-dimensional environments where the mesh requires considerable

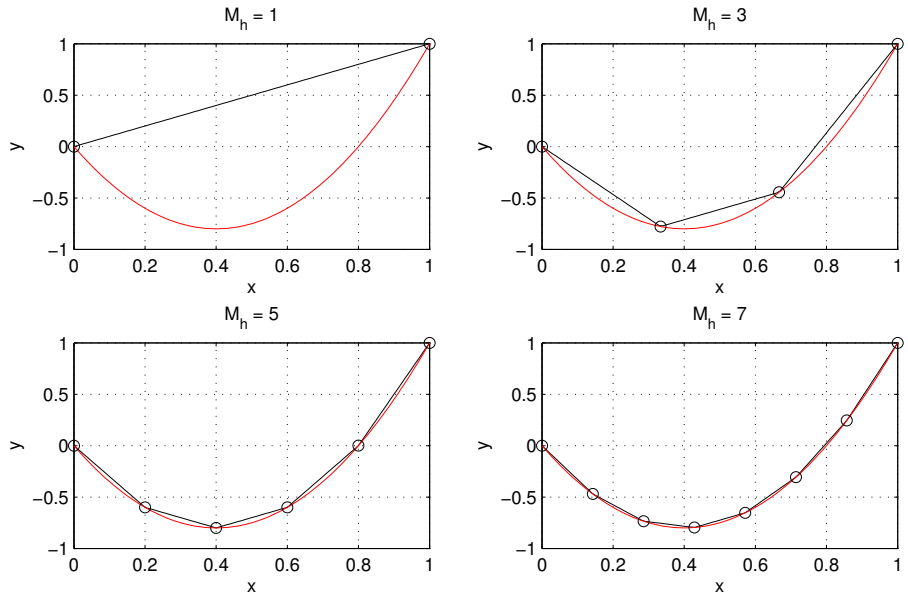

Figure 2.4: h-refinement of a one-dimensional mesh. It can be seen how the refinement of the mesh produces a more precise result (black curve).

computational load. Anyway, this kind of refinement is considered to be superior.

3. Another kind of refinement is the *r-refinement*: this implies only a repositioning of the nodes already existing.

For what concerns p-refinement, we can recognize two different approaches:

1. uniform increase of the polynomial order on every elements;
2. local increase of the polynomial order typically using hierarchical refinement.

It is furthermore possible to combine the positive aspects of both the refinement types, getting what's called *hp-refinement*. In this kind of procedure, both the degree of the element and its size are refined in order to produce an approximation nearer to the exact solution.

2.10.1 Prediction of the element size

In practical applications, we commonly try to find an approximation whose relative energy norm percentage error ϵ is less than a specified $\bar{\epsilon}$, specifically defined for each application. The ideal case is the one in which the distribution of energy norm $\|\boldsymbol{e}\|_e$ (see A.20) is uniform on all elements. Thus, the permissible error is

$$PE = \bar{\epsilon}\,\|\boldsymbol{u}\|_e = \bar{\epsilon}\sqrt{\|\hat{\boldsymbol{u}}\|_e^2 + \|\boldsymbol{e}\|_e^2},$$

where

$$\|\boldsymbol{e}\|_e^2 = \|\boldsymbol{u}\|_e^2 - \|\hat{\boldsymbol{u}}\|_e^2\,.$$

As already stated, we could require that the error is distributed equally on each element, so that we can require that the error for the k^{th} element is

$$\|\boldsymbol{e}\|_{e,k} < \bar{\epsilon}\sqrt{\frac{\|\hat{\boldsymbol{u}}\|_e^2 + \|\boldsymbol{e}\|_e^2}{M_h}} \equiv \bar{e}_m, \tag{2.33}$$

where M_h is the total number of elements in the mesh and $\bar{e}_m$ is the error energy required for each element. The elements for which Equation (2.33) are candidates for refinement. This means we can refine those elements only, reducing their size. In this case, the technique of mesh subdivision is employed, but its efficiency is dependant on the fact that, despite the reasonable number of degrees of freedom, the number of trial solutions may be excessive.

It can be shown a more efficient way of refining basing on the energy error is performing a complete remeshing, creating a new mesh which satisfies the condition of Equation (2.33) for all the elements k in the mesh. We can suppose, for instance,

$$\|\boldsymbol{e}\|_k \propto h_k^p$$

where h_k is the size of the k^{th} element and p is the order of the polynomial used in the approximation. This means the new element size should be no larger than

$$\bar{h}_k = \left(\frac{\|\boldsymbol{e}\|_k}{\bar{e}_m}\right)^{-\frac{1}{p}} h_k.$$

2.10.2 *p*- and *hp*-refinement

Nonuniform p-refinement is possible, and it can be done hierarchically. However, generalizing the process is difficult and many assumptions are needed about the decrease of the error. More information about this can be found in [31, 34].

hp-refinement is very interesting and recent works have proved it to be an efficient technique of refining. An efficient methodology has been proposed in [39, 34]. The first step requires to pursue through h-refinement with lowest-order elements an accuracy around 5% with uniform energy norm error. After this, a p-refinement is performed uniformly on the elements. The result is an efficient procedure with easy implementation.

Relevant in this regard is the *HERMES project*: Hermes is a free C++/Python library for rapid prototyping of adaptive FEM and *hp*-FEM solvers developed by an open source community around the *hp*-FEM group at the University of Nevada, Reno. The library has a clean design and modular structure, and it is available under the GPL license (Version 2, 1991).

Chapter 3

Bézier, B-spline, NURBS and T-spline

The mathematical representation of curves, surfaces and solids is fundamental in a design process. It will be shown in Chapter 4 that these structures will be used in the analysis as well, making these technologies with their properties key concepts in a production environment.

3.1 Analytical representation

The most common representation forms of a curve or of a surface are the implicit and the parametric form.

In the implicit form, the equation is given in a way in which the dependent variable is not given explicitly in terms of the independent variables. The general form of an implicit equation is

$$f(x_1, x_2, \ldots, x_n, y) = 0.$$

A surface lying on the xy plane is then written

$$f(x, y) = 0.$$

By contrast, the explicit representation is given so that the dependent variable is explicitly given as a function of the independent variables:

$$y = f(x_1, x_x, \ldots, x_n).$$

A different way of representing an equation is the parametric form

$$\boldsymbol{F}(\boldsymbol{\xi}) = (g_1(\boldsymbol{\xi}), g_2(\boldsymbol{\xi}), \ldots, g_n(\boldsymbol{\xi})), \ \boldsymbol{c} \leq \boldsymbol{\xi} \leq \boldsymbol{d}.$$

Each coordinate of an element in $\mathrm{im}(\boldsymbol{F})$ is represented separately as an explicit function of an independent parameter.

Example 3.1. The ellipse in Figure 3.1a can be expressed with the implicit form

$$\frac{x^2}{3.5^2} + \frac{y^2}{2^2} = 1 \tag{3.1}$$

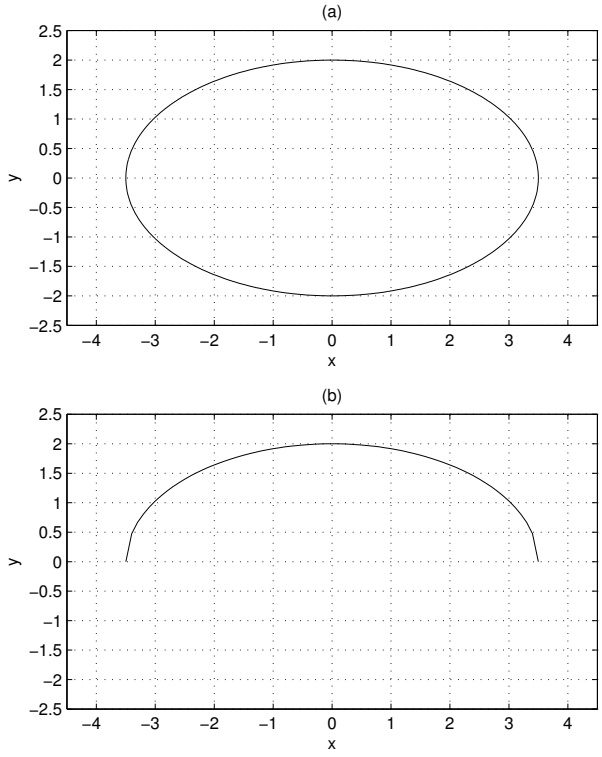

Figure 3.1: (a) Ellipse on the xy plane defined using a parametric or an implicit form. (b) Ellipse on the xy plane defined using an explicit form.

	Implicit form	Parametric form
Curve	$c(x,y)=0$	$\boldsymbol{C}(\xi)=(x(\xi),y(\eta))$
Surface	$s(x,y,z)=0$	$\boldsymbol{S}(\xi,\eta)=(x(\xi,\eta),y(\xi,\eta),z(\xi,\eta))$

Table 3.1: Comparison between parametric and implicit forms of curves on the xy plane and surfaces in the xyz space.

or with the parametric form

$$\boldsymbol{C}(\xi)=(x(\xi),y(\xi))=(4\cdot\cos(\xi),2\cdot\sin(\xi)).$$

The explicit form, instead, cannot express the entire ellipse as it doesn't fit the definition of function. Solving Equation 3.1 by y we get

$$y(x)=\pm\frac{2}{3.5}\left(-x^2+3.5^2\right)^{\frac{1}{2}}.$$

The positive part is plotted in Figure 3.1b.

A widely used scheme for writing surfaces in the xyz space is the *tensor product* scheme. This scheme can be expressed through the analytic form

$$\boldsymbol{S}(\xi,\eta)=\sum_{i=0}^{n}\sum_{j=0}^{m}f_i(\xi)\,g_j(\eta)\,\boldsymbol{b}_{i,j},\ \boldsymbol{a}\leq\boldsymbol{\xi}\leq\boldsymbol{b},$$

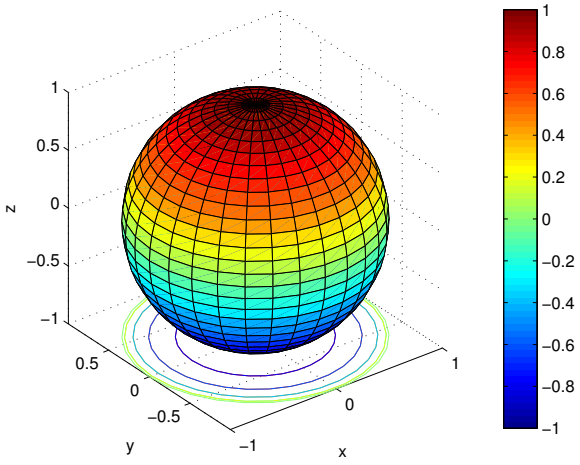

Figure 3.2: Sphere in the xyz space.

where

$$\boldsymbol{b}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})\,.$$

However, it is possible to express a surface again both with the implicit form

$$f(x, y, z) = 0,$$

and with the explicit form

$$z = f(x, y)\,.$$

Example 3.2. The sphere in Figure 3.2 can be expressed by the implicit form

$$x^2 + y^2 + z^2 - 1 = 0$$

or with the parametric form

$$\boldsymbol{S}(\xi, \eta) = (\sin\xi \cdot \cos\eta, \sin\xi \cdot \sin\eta, \cos\xi)\,,\ 0 \le \xi \le \pi,\ 0 \le \eta \le 2\pi.$$

3.2 Power basis curves and surfaces

It can be seen from the general form of a parametric curve in Table 3.1 that the class of curves definable is very large. Anyway, such a general form presents some difficulties which suggest the need for a restriction in order to be more practical. A good class of functions to use is the class of the polynomials.

Hence, the general form of a n^{th}-degree power basis curve is:

$$\begin{aligned}\boldsymbol{C}(\xi) &= (x(\xi), y(\xi), z(\xi)) \\ &= \sum_{i=0}^{n} \boldsymbol{a}_i \xi^i \\ &= \left([\boldsymbol{a}_i]_{i=0}^{n}\right)^T \left[\xi^i\right]_{i=0}^{n},\end{aligned}$$

with $b \leq \xi \leq c$. The $\boldsymbol{a}_i$'s are the row vectors (x_i, y_i, z_i) and the $n+1$ functions ξ^i are called *basis* (or *blending*) *functions*.

Polynomials are a good choice as they're efficiently represented and managed by computers and are mathematically simple to handle. Unfortunately polynomials are not capable of representing many kinds of important curves. In these cases it is necessary to approximate.

Following the tensor product scheme, it is simple to guess that a power basis surface can be represented in a similar way:

$$\begin{aligned} \boldsymbol{S}(\xi,\eta) &= (x(\xi,\eta), y(\xi,\eta), z(\xi,\eta)) \\ &= \sum_{i=0}^{n}\sum_{j=0}^{m} \boldsymbol{a}_{i,j}\xi^i\eta^j \\ &= \left(\left[\xi^i\right]_{i=0}^{n}\right)^T \left[\boldsymbol{a}_{i,j}\right]_{i,j=0}^{i=n,j=m} \left[\eta^j\right]_{j=0}^{m} \end{aligned}$$

where

$$\begin{cases} \boldsymbol{a}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \\ \boldsymbol{b} \leq \boldsymbol{\xi} \leq \boldsymbol{c} \end{cases}.$$

3.3 Bézier curves and surfaces

Bézier curves use polynomials like the power basis representation. This fact makes the two representations equivalent, where by equivalent it means any curve representable with the power basis form is representable even with the Bézier form. However, the power basis representation shows three disadvantages:

- the coefficients $\boldsymbol{a}_i$ convey almost no geometrical meaning to the user;
- algorithms for processing power basis curves have a more algebraic connotation rather than geometrical;
- more prone to round-off error.

Bézier curves improve the power basis representation relieving these problems.

An n^{th}-degree Bézier curve is defined by

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} B_i^n(\xi)\,\boldsymbol{P}_i,\ a \leq \xi \leq b.$$

The functions

$$B_i^n(\xi) = \frac{n! \cdot \xi^i (1-\xi)^{n-i}}{i! \cdot (n-i)!}$$

are the *basis functions* (n^{th}-degree *Bernstein polynomial*), similarly to the terms ξ^i of the power basis representation, and the $\boldsymbol{P}_i$'s are called *control points*.

Example 3.3. The Bézier curve in two dimensions given in the figure 3.3a, for instance, can be plotted using the Bernstein polynomials represented in the figure 3.3b, using the control points

$$\boldsymbol{P}_0 = (0,0),\ \boldsymbol{P}_1 = (1,1),\ \boldsymbol{P}_2 = (2,0.5),$$

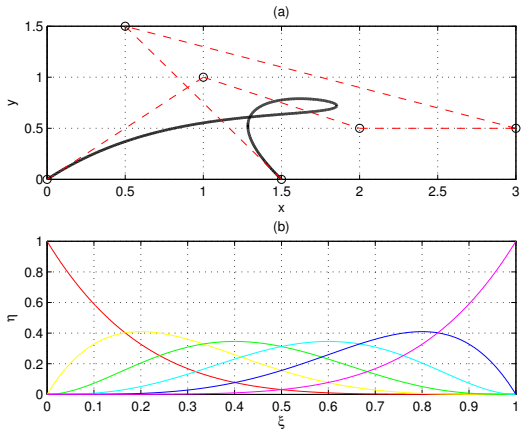

Figure 3.3: Example of (a) Bézier curve on the xy plane with (b) its Bernstein polynomials.

$$\boldsymbol{P}_3 = (3, 0.5)\,,\ \boldsymbol{P}_4 = (0.5, 1.5)\,,\ \boldsymbol{P}_5 = (1.5, 0)\,.$$

If, instead, the control points are given in a three-dimensional space

$$\boldsymbol{P}_0 = (0, 0, 0)\,,\ \boldsymbol{P}_1 = (1, 1, 1)\,,\ \boldsymbol{P}_2 = (2, 0.5, 0)\,,$$

$$\boldsymbol{P}_3 = (3, 0.5, 0)\,,\ \boldsymbol{P}_4 = (0.5, 1.5, 0)\,,\ \boldsymbol{P}_5 = (1.5, 0, 1)\,,$$

then a three-dimensional Bézier curve is generated (see Figure 3.4).

A nonrational Bézier surface is obtained by using the expression

$$\boldsymbol{S}(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(\xi)\, B_j^m(\eta)\, \boldsymbol{P}_{i,j},\ \begin{cases} a \le \xi \le b \\ c \le \eta \le d \end{cases}.$$

The control points $\boldsymbol{P}_{i,j}$ form a bidirectional net like the one illustrated in the example Figure 3.5.

3.3.1 Rational Bézier curves and surfaces

There is a considerable amount of important shapes which cannot be represented with polynomials only. Curves or surfaces such as circles, cones, ellipses, spheres, etc..., for instance, can be represented with rational functions, which are ratios of polynomials:

$$x(\xi) = \frac{\chi(\xi)}{w(\xi)}, \quad y(\xi) = \frac{\upsilon(\xi)}{w(\xi)}.$$

An n^{th}-degree rational Bézier curve is defined

$$\boldsymbol{C}(\xi) = \frac{\sum_{i=0}^{n} B_i^n w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} B_i^n(\xi)\, w_i},\ a \le \xi \le b, \tag{3.2}$$

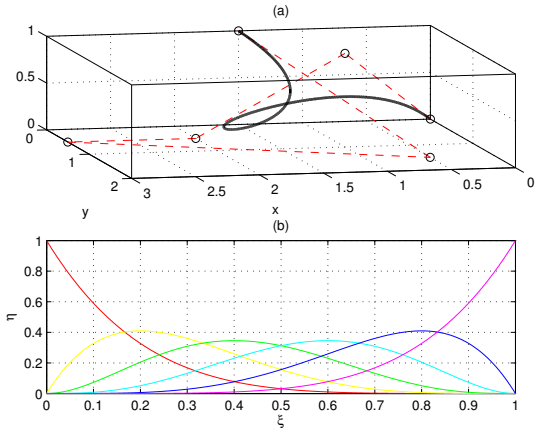

Figure 3.4: Example of (a) Bézier curve in the xyz space with (b) its Bernstein polynomials.

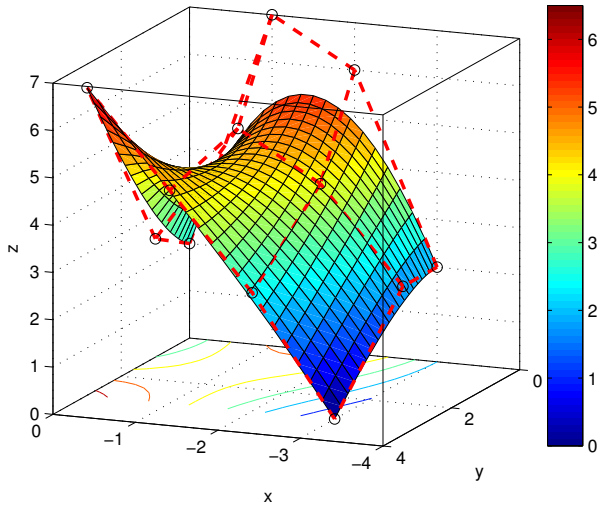

Figure 3.5: Example of a Bézier surface in the xyz space define by its control points.

where the w_i's are scalars called *weights*. It is possible to rewrite the definition (3.2) using the notation

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} R_i^n(\xi)\,\boldsymbol{P}_i,\ a \le \xi \le b,$$

where

$$R_i^n(\xi) = \frac{B_i^n(\xi)\,w_i}{\sum_{\hat{i}=0}^{n} B_{\hat{i}}^n(\xi)\,w_{\hat{i}}}$$

are called *rational basis functions*.

It is possible to change again the representation of the rational Bézier curves using a new interpretation which yields efficient processing and storage. The usage of homogeneous coordinates permits to represent a rational curve in an n-dimensional space as a curve in $n+1$ dimensions. If $\boldsymbol{P} = (x, y, z)$ is a point in a 3-dimensional space, it can be represented as $\boldsymbol{P}^w = (wx, wy, wz, w)$ in a 4-dimensional space. A function H is defined as a mapping of a point

$$\boldsymbol{P} = H(\boldsymbol{P}^w) = H((x, y, z, w)) = \begin{cases} \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) & w \neq 0 \\ (x, y, z) & w = 0 \end{cases}$$

to the hyperplane $w = 1$. Employing this concept, a new definition of a rational Bézier curve in a 3-dimensional space can be the nonrational Bézier curve

$$\boldsymbol{C}^w(\xi) = \sum_{i=0}^{n} B_i^n(\xi)\,\boldsymbol{P}_i^{w_i}$$

defined in a 4-dimensional space. Applying the transformation H to $\boldsymbol{C}^w(\xi)$ we get the nonrational Bézier curve as the projection of the rational curve on the plane $w = 1$.

Example 3.4. Consider we want to represent the 2-dimensional rational Bézier curve with the control points

$$\boldsymbol{P}_0 = (1, 0),\ \boldsymbol{P}_1 = (1, 1),\ \boldsymbol{P}_2 = (0, 1),$$

and with the weights

$$w_0 = 1,\ w_1 = 1,\ w_2 = 2.$$

It is possible to write the curve

$$\boldsymbol{C}(\xi) = \frac{\sum_{i=0}^{2} B_i^n(\xi)\,w_i\boldsymbol{P}_i}{\sum_{i=0}^{2} B_i^n(\xi)\,w_i}$$

or using the rational basis functions:

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{2} R_i^n(\xi)\,\boldsymbol{P}_i,\ R_i^n(\xi) = \frac{B_i^n(\xi)\,w_i}{\sum_{\hat{i}=0}^{2} B_{\hat{i}}^n(\xi)\,w_{\hat{i}}}.$$

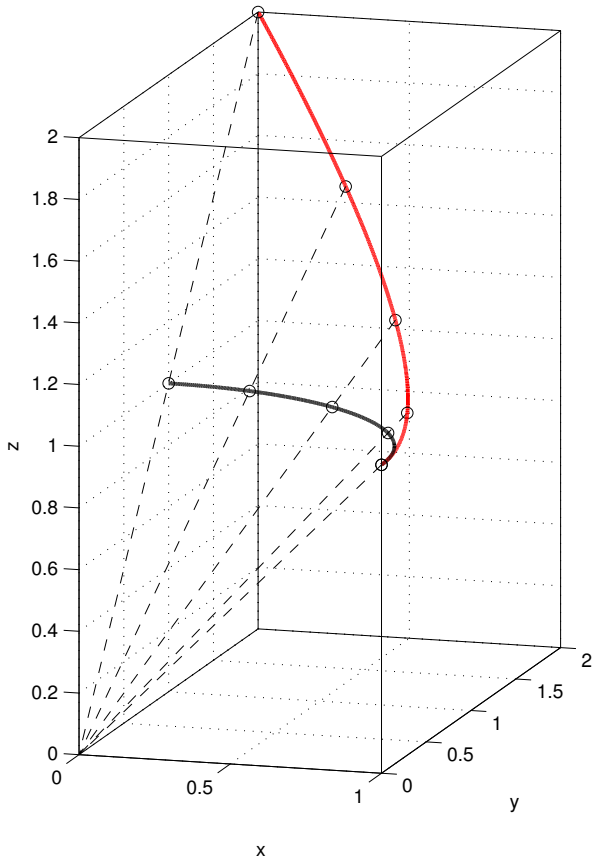

Figure 3.6: Example of mapping of a nonrational Bézier curve (black curve in figure) to a rational Bézier curve (red curve in figure).

In both cases the representation is the one of the black curve of the figure 3.6. It is possible, as stated above, to use a better notation, moving to a higher dimension. The control points $\boldsymbol{P}_i = (x_i, y_i)$ needs to be redefined to $\boldsymbol{P}_i^{w_i} = (w_i x_i, w_i y_i, w_i)$. This way, the new curve can be represented in the 3-dimensional space like the red curve in the figure 3.6.

A rational Bézier surface can be written

$$\boldsymbol{S}(\xi,\eta) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(\xi)\, B_j^m(\eta)\, w_{i,j}\boldsymbol{P}_{i,j}^w}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(\xi)\, B_j^m(\eta)\, w_{i,j}}, \quad \begin{cases} a \le \xi \le b \\ c \le \eta \le d \end{cases}$$

or using a nonrational expression

$$\boldsymbol{S}^w(\xi,\eta) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(\xi)\, B_j^m(\eta)\, \boldsymbol{P}_{i,j}^w, \quad \begin{cases} a \le \xi \le b \\ c \le \eta \le d \end{cases}.$$

3.4 Univariate and multivariate B-splines

Both Bézier and power basis curves are affected by some shortcomings:

- a high degree polynomial is needed to satisfy many constraints;
- a high degree is required to approximate some shapes;
- local control of the curve is difficult.

A solution to these points is to employ piecewise-polynomials or piecewise-rational polynomials. The idea is to construct the curves dividing the domain and associating each part of the domain with a distinct polynomial. Points separating different parts of the domain are called *breakpoints*. Each segment of the B-spline curve connects to another on the breakpoint with some level of continuity: a curve $\boldsymbol{C}(\xi)$ is said to be C^k continuous at the breakpoint ξ_i if $C_i^{(j)}(\xi_i) = C_{i+1}^{(j)}(\xi_i)$, $\forall\, 0 \leq j \leq k$.

A good choice of B-spline basis functions is that discussed in the subsection 3.4.1.

3.4.1 B-spline basis functions

Given is a set $\Xi = [\xi_o, \xi_1, \ldots, \xi_m]$ where $\xi_i \in \mathbb{R},\ i = 0, \ldots, m$ and $\xi_i \leq \xi_{i+1}$, $i = 0, \ldots, m-1$. The ξ_i's are called *knots* and Ξ is called *knot vector*. The i^{th} B-spline basis function of degree p (and order $p+1$) is

$$N_i^0(\xi) = \begin{cases} 1, & \xi_i \leq \xi < \xi_{i+1} \\ 0, & \text{otherwise} \end{cases},$$

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} \cdot N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} \cdot N_{i+1}^{p-1}(\xi). \tag{3.3}$$

Definition 3.5. A knot vector of the form

$$\Xi = \left[\underbrace{a, \ldots, a}_{p+1}, \xi_{p+1}, \ldots, \xi_{m-p-1}, \underbrace{b, \ldots, b}_{p+1} \right]$$

where p is the degree and m is the number of elements of the knot vector, is said to be *nonperiodic* or *clamped* or *open*.

Example 3.6. In Figure 3.7 B-spline basis functions over the knot vector $\Xi = [0, \ldots, 0, 1, 4, 6, 8, \ldots, 8]$ of increasing degree are represented.

Property1: This choice of the basis functions guarantees the *local support property*: $N_i^p(\xi) = 0$ if $\xi \notin [\xi_i, \xi_{i+p+1})$. This fact can be simply seen in Figure 3.7.

Property2: In any given knot span $[\xi_j, \xi_{j+1})$ at most $p+1$ of the N_i^p are nonzero, namely the functions $N_{j-p}^p, \ldots, N_j^p$.

Property3: The property of *nonnegtivity* states that $N_i^p(\xi) \geq 0$ for all i, p and ξ.

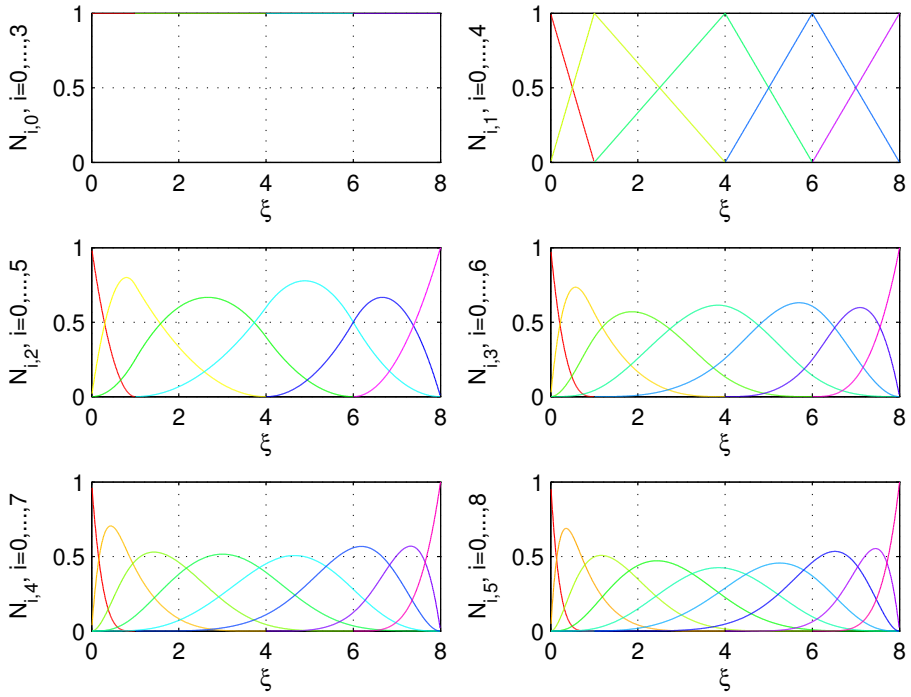

Figure 3.7: Example of B-spline basis functions over the knot vector $\Xi = \{0, ..., 0, 1, 4, 6, 8, ..., 8\}$ of increasing degree.

Property4: The property of partition of unity states that for any arbitrary knot span $[\xi_i, \xi_{i+1})$, $\sum_{j=i-p}^{i} N_j^p(\xi) = 1$ for all $\xi \in [\xi_i, \xi_{i+1})$.

Property5: $N_i^p(\xi)$ is $C^\infty((\xi_j, \xi_{j+1}))$ for all j. At a knot $N_i^p(\xi)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot.

Property6: The set of all B-spline basis functions of p^{th}-degree $N_i^p(\xi)$, $i = 0, \ldots, n$ defined on the knot vector

$$\Xi = \left[\underbrace{\xi_0, \ldots, \xi_0}_{s_0}, \underbrace{\xi_1, \ldots, \xi_1}_{s_1}, \ldots, \underbrace{\xi_k, \ldots, \xi_k}_{s_k}\right]$$

forms a *basis of the space* $\mathcal{S}_\Xi^p$ *of the of the piecewise-polynomials of degree* p *with continuity* C^{r_j} *at* ξ_j *with* $r_j = p - s_j$. It is possible to show that the dimension of the space $\mathcal{S}_\Xi^p$ is

$$\dim(\mathcal{S}_\Xi^p) = k(p+1) - \sum_{j=0}^{k}(r_j + 1). \tag{3.4}$$

3.4.2 Algorithms for B-spline basis functions

A frequent need when working with B-spline basis functions is to compute all the nonvanishing functions in a specific point ξ. Basing on the equations (3.3), suppose we want to compute all the nonvanishing functions in $\xi \in [\xi_i, \xi_{i+1})$ when $p = 2$, we get:

$$N_{i-2}^2(\xi) = \frac{\xi - \xi_{i-2}}{\xi_i - \xi_{i-2}} N_{i-2}^1(\xi) + \frac{\xi_{i+1} - \xi}{\xi_{i+1} - \xi_{i-1}} \underbrace{N_{i-1}^1}_{(*)}(\xi),$$

$$N_{i-1}^2(\xi) = \frac{\xi - \xi_{i-1}}{\xi_{i+1} - \xi_{i-1}} \underbrace{N_{i-1}^1}_{(*)}(\xi) + \frac{\xi_{i+2} - \xi}{\xi_{i+2} - \xi_i} \underbrace{N_i^1}_{(**)}(\xi),$$

$$N_i^2(\xi) = \frac{\xi - \xi_i}{\xi_{i+2} - \xi_i} \underbrace{N_i^1}_{(**)}(\xi) + \frac{\xi_{i+3} - \xi}{\xi_{i+3} - \xi_{i+1}} N_{i+1}^1(\xi).$$

The asterisks underline the presence, in different basis functions, of the same functions. It is possible, therefore, to reuse the same value, instead of calculating it more than once. Algorithm 3.2 considers this improvement and computes all the functions which are not zero in the provided point ξ. As a conceptual scheme, what we have to compute is the inverted triangular table

$$\begin{array}{ccccc} & & & & N_{i-p}^p \\ N_i^0 & N_{i-1}^1 & \cdots & & N_{i-p+1}^p \\ & N_i^1 & & & \vdots \\ & & & & N_i^p \end{array}. \tag{3.5}$$

Algorithm 3.1 Algorithm for the determination of the knot span in which the provided value lies.

```
% findSpan finds in which knot span a specific
% value of xi is to be found.
% Input:
%   n: max index of the control points (n+1 control points);
%   p: degree of the B-spline basis functions;
%   xi: scalar value to be found;
%   Xi: open knot vector {xi_0,...,xi_n,...,x_{n+p+1}}.
% Output:
%   i: knot span [xi_i, xi_{i+1}) in which u lies.
function i = findSpan(n, p, xi, Xi)
% Special case.
if xi == Xi(n+2), i = n; return; end;
% Binary search.
low = p;
high = n+1;
i = floor((low + high)./2);
while xi < Xi(i+1) || xi >= Xi(i+2)
    if xi < Xi(i+1); high = i;
    else low = i; end;
    i = floor((low + high)./2);
end
```

Algorithm 3.2 Algorithm for the evaluation of every nonvanishing B-spline basis function in the provided value.

```
% Compute the nonvanishing basis functions.
% Input:
%   i: index of the basis function to compute (this value cannot be
    smaller
%       than p);
%   xi: value in which the basis function is being evaluated;
%   p: degree of the basis function;
%   Xi: knot vector over which the basis function is being built.
% Output:
%   N: vector containing the value of all the nonvanishing basis
    functions:
%   N(1)=N_{i-p},...,N(p+1)=N_{i}.
function N = basisFuns(i, xi, p, Xi)
% Preallocation.
N = zeros(1, p+1);
left = zeros(1, p+1);
right = zeros(1, p+1);
% Computation of the inverse triangular table starting from degree 0.
N(1) = 1;
for j = 1:p
    left(j+1) = xi-Xi(i+1-j+1);
    right(j+1) = Xi(i+j+1)-xi;
    saved = 0;
    for r = 0:j-1
        temp = N(r+1)./(right(r+1+1)+left(j-r+1));
        N(r+1) = saved+right(r+1+1).*temp;
        saved = left(j-r+1).*temp;
    end
    N(j+1) = saved;
end
```

Algorithm 3.3 Algorithm for the computation of the i^{th} B-spline basis function in ξ.

```
% Evaluates the value of the i-th B-spline basis function of degree p over
% the knot vector Xi in xi.
% Input:
%   p: degree of the function to evaluate;
%   m: number of knots - 1;
%   Xi: knot vector over which the basis function is built;
%   i: index of the B-spline basis function to compute;
%   xi: point where to evaluate the B-spline basis function.
% Output:
%   Nip: value of the B-spline basis function in xi.
function Nip = basisFun(p, m, Xi, i, xi)
% Check to see if we're evaluating the first or the last basis function at
% the beginning or at the end of the knot vector.
if (i == 0 && xi == Xi(0+1)) || (i == m-p-1 && xi == Xi(m+1))
    Nip = 1; return;
end
% When xi is out of the domain it is set to zero.
if (xi < Xi(i+1) || xi >= Xi(i+p+1+1))
    Nip = 0; return;
end
% Preallocation and computation of the temparary values of the functions to
% be used according to the triangular table.
N = zeros(p+1);
for j = 0:p
    if xi(1) >= Xi(i+j+1) && xi(1) < Xi(i+j+1+1), N(j+1) = 1;
    else N(j+1) = 0; end;
end
% Computation of the rest of the triangular table.
for k = 1:p
    if N(1) == 0, saved = 0;
    else saved = ((xi(1)-Xi(i+1)).*N(0+1))./(Xi(i+k+1)-Xi(i+1)); end;
    for j = 0:p-k-1-1
        Xileft = Xi(i+j+1+1);
        Xiright = Xi(i+j+k+1+1);
        if N(j+1+1) == 0
            N(j+1) = saved;
            saved = 0;
        else
            temp = N(j+1+1)./(Xiright-Xileft);
            N(j+1) = saved+(Xiright-xi(1)).*temp;
            saved = (xi(1)-Xileft).*temp;
        end
    end
end
Nip = N(1);
```

In case we only want to compute a single B-spline basis function, we can try to produce an algorithm considering which basis functions are needed and which are not. We can come up with this triangular table:

$$\begin{array}{cccc} N_i^0 & & & \\ N_{i+1}^0 & N_i^1 & & \\ \vdots & \vdots & \cdots & N_i^{p-1} \quad N_i^p. \\ N_{i+p-1}^0 & N_{i+p-1}^1 & & N_{i+1}^{p-1} \\ N_{i+p}^0 & & & \end{array} \tag{3.6}$$

These are the only functions needed when computing N_i^p, and the procedure in Algorithm 3.3 computes this table, beginning from the degree 0.

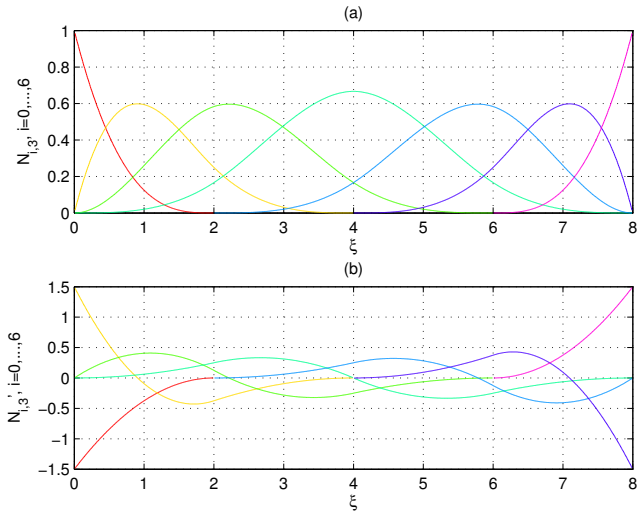

Figure 3.8: Example of derivatives of the basis functions of Example 3.6 with $p = 3$.

3.4.3 Algorithms for B-spline basis functions derivatives

The derivative of a B-spline basis function $N_i^p(\xi)$ can be computed with

$$\frac{\partial N_{i,p}(\xi)}{\partial \xi} = \frac{p}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi). \tag{3.7}$$

Example 3.7. Using the recursive definition (3.7) on the B-spline basis functions of degree $p = 3$ of Example 3.6 it is possible to obtain the curves of Figure 3.8.

Example 3.8. Using the recursive definition (3.7) on the B-spline basis functions of degree $p = 3$ built over the knot vector

$$\Xi = \{0, 0, 0, 0, 2, 4, 4, 6, 6, 6, 8, 8, 8, 8\}$$

it is possible to obtain the curves of Figure 3.9. These curves shows the behaviour in case of single, double and triple knots.

The derivatives of the B-spline basis functions are important as they lead to another important property:

Property7 In the interior of a knot span, all the derivatives of $N_i^p(\xi)$ exist (as it is a polynomial); at a knot it is $p - k$ times continuously differentiable, where k is the multiplicity of the knot. This means that increasing the degree increases the continuity whereas increasing the knot multiplicity decreases the continuity.

Two efficient algorithms for the determination of the derivatives of B-spline basis functions are given: Algorithm 3.4 computes the derivatives of order $0 \leq k \leq n$ of all the nonvanishing functions in the given value; Algorithm 3.6 determines the derivatives of order $0 \leq k \leq n$ of the i^{th} function.

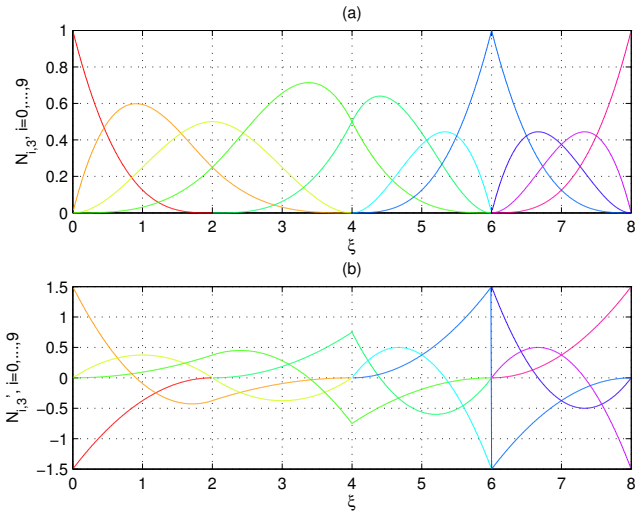

Figure 3.9: Example of derivatives of the B-spline basis functions of degree $p = 3$ built over the knot vector $\Xi = \{0, 0, 0, 0, 2, 4, 4, 6, 6, 6, 8, 8, 8, 8\}$.

Algorithm 3.4 Algorithm for the determination of the derivatives of order $0 \leq k \leq n$ of all the nonvanishing B-spline basis functions in the specified ξ (continues to Algorithm 3.5).

```
% derivsBasisFuns computes the nonzero basis functions and
% their derivatives.
% Input:
%   i: index of the basis function to compute (i=0,1,...,n).
%   xi: point where the derivatives are evaluated;
%   p: degree of the basis functions;
%   n: defined accordingly to the knot vector;
%   Xi: knot vector.
% Output:
%   ders: bidimensional matrix where the element in position
%         (k,j) is the (k-1)-th derivative of the function
%         N_{i-p+j,p} with 0<=k<=n and 0<=j<=p.
function ders = derivsBasisFuns(i, xi, p, n, Xi)
% First evaluate the basis functions.
ndu(1, 1) = 1;
left(p+1) = 0;
right(p+1) = 0;
for j=1:p
    left(j+1) = xi-Xi(i+1-j+1);
    right(j+1) = Xi(i+j+1)-xi;
    saved = 0;
    for r=0:j-1
        ndu(j+1, r+1) = right(r+2)+left(j-r+1);
        temp = ndu(r+1, j)./ndu(j+1, r+1);
        ndu(r+1, j+1) = saved+right(r+2).*temp;
        saved = left(j-r+1).*temp;
    end
    ndu(j+1, j+1) = saved;
end
% Load the basis functions.
ders(n+1, p+1) = 0;
for j=1:p+1
    ders(1, j) = ndu(j, p+1);
end
% (continues...)
```

Algorithm 3.5 Algorithm for the determination of the derivatives of order $0 \leq k \leq n$ of the i^{th} B-spline basis function in the specified ξ (continues from Algorithm 3.4).

```
% (...continues)
% Evaluation of the derivatives.
% Loop over function index.
a(p, p) = 0;
for r=0:p
    % Indices for the matrix containing the derivatives.
    s1 = 0; s2 = 1;
    a(1, 1) = 1;
    % Loop for the computation of the kth derivative.
    for k=1:n
        d = 0;
        rk = r-k; pk = p-k;
        if r >= k
            a(s2+1, 0+1) = a(s1+1, 0+1)./ndu(pk+2, rk+1);
            d = a(s2+1, 0+1).*ndu(rk+1, pk+1);
        end
        if rk >= -1, j1 = 1;
        else j1 = -rk; end;
        if r-1 <= pk, j2 = k-1;
        else j2 = p-r; end;
        for j=j1:j2
            a(s2+1,j+1) =...
                (a(s1+1, j+1)-a(s1+1, j-1+1))./ndu(pk+1+1, rk+2);
            d = d+a(s2+1, j+1).*ndu(rk+j+1, pk+1);
        end
        if r <= pk
            a(s2+1, k+1) = -a(s1+1, k)./ndu(pk+2, r+1);
            d = d+a(s2+1, k+1).*ndu(r+1, pk+1);
        end
        ders(k+1, r+1) = d;
        j=s1; s1 = s2; s2 = j;
    end
end
% Multiply by the correct factors.
r = p;
for k=1:n
    for j=0:p
        ders(k+1, j+1) = ders(k+1, j+1).*r;
    end
    r = r.*(p-k);
end
```

$N_i^0(\xi)$	$N_{i-1}^1(\xi)$	$N_{i-2}^2(\xi)$
$\xi_{i+1} - \xi_i$	$N_i^1(\xi)$	$N_{i-1}^2(\xi)$
$\xi_{i+1} - \xi_{i-1}$	$\xi_{i+2} - \xi_i$	$N_i^2(\xi)$

Table 3.2: Scheme of structure `ndu(i,j)` used in Algorithm 3.4 to store the elements necessary for the computation.

Algorithm 3.4 is based on a generalization of Equation (3.7):

$$N_i^{p,(k)}(\xi) = \frac{p!}{(p-k)!} \sum_{j=0}^{k} a_{k,j} N_{i+j}^{p-k}(\xi),$$

where

$$\begin{aligned} a_{0,0} &= 1, \\ a_{k,0} &= \frac{a_{k-1,0}}{\xi_{i+p-k+1} - \xi_i}, \\ a_{k,j} &= \frac{a_{k-1,j} - a_{k-1,j-1}}{\xi_{i+p+j-k+1} - \xi_{i+j}},\ j = 1, \ldots, k-1, \\ a_{k,k} &= \frac{-a_{k-1,k-1}}{\xi_{i+p+1} - \xi_{i+k}}. \end{aligned}$$

By analyzing this equation, it is simple to see that we need the inverted triangular table, the differences of knots and the differences of the $a_{k,j}$'s. A possible structure to maintain these data is a table like Table 3.2. In Table 3.2 the necessary data to compute the $a_{k,j}$'s and then the $N_i^{p,(k)}$'s is present.

Algorithm 3.6 is based, instead, on the Equation

$$N_i^{p,(k)}(\xi) = p \left(\frac{N_i^{p-1,(k-1)}(\xi)}{\xi_{i+p} - \xi_i} - \frac{N_{i+1}^{p-1,(k-1)}(\xi)}{\xi_{i+p+1} - \xi_{i+1}} \right), \tag{3.8}$$

which can be derived by repeated differentiation of Equation (3.7). By using Equation (3.8), for instance, for $p = 3$ and $k = 0, \ldots, n$, with $n = 3$ we get:

$$\begin{aligned} N_i^{3,(1)}(\xi) &= 3 \left(\frac{N_i^2(\xi)}{\xi_{i+3} - \xi_i} - \frac{N_{i+1}^2}{\xi_{i+4} - \xi_{i+1}} \right), \\ N_i^{3,(2)}(\xi) &= 3 \left(\frac{N_i^{2,(1)}(\xi)}{\xi_{i+3} - \xi_i} - \frac{N_{i+1}^{2,(1)}}{\xi_{i+4} - \xi_{i+1}} \right), \\ N_i^{3,(3)}(\xi) &= 3 \left(\frac{N_i^{2,(2)}(\xi)}{\xi_{i+3} - \xi_i} - \frac{N_{i+1}^{2,(2)}}{\xi_{i+4} - \xi_{i+1}} \right). \end{aligned}$$

Now, we can use some tables to compute all and only the values needed to get all the derivatives up to degree 3 of the basis functions. The first computes the i^{th} basis function of degree 3 (the triangular table of (3.6)):

$$\begin{array}{cccc} N_i^0 & & & \\ & N_i^1 & & \\ N_{i+1}^0 & & N_i^2 & \\ & N_{i+1}^1 & & N_i^3. \\ N_{i+2}^0 & & N_{i+1}^2 & \\ & N_{i+2}^1 & & \\ N_{i+3}^3 & & & \end{array}$$

Then, the k^{th} derivative, $0 \leq k \leq n$, can be computed with a table of this kind:

$$\begin{matrix} N_i^{p-k} & & \\ \vdots & \cdots & \begin{matrix} N_i^{2,(k-1)} \\ N_{i+1}^{2,(k-1)} \end{matrix} \quad N_i^{3,(k)}. \\ N_{i+k}^{p-k} & & \end{matrix}$$

3.4.4 Univariate B-splines

The general concept of the B-spline curves is to maintain the same structure used to define the Bézier curves or the power basis curves, designing the basis functions according to the idea exposed above:

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} f_i(\xi)\, \boldsymbol{P}_i,\ a \leq \xi \leq b.$$

As already stated, a good choice for the $f_i(\xi)$'s are the basis functions defined in Subsection 3.4.1; this leads to the form

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} N_i^p(\xi)\, \boldsymbol{P}_i,\ a \leq \xi \leq b,$$

where the $\boldsymbol{P}_i$'s are the control points, and the $N_i^p(\xi)$'s are the p^{th}-degree B-spline basis functions of the equation (3.3) defined on the nonuniform knot vector

$$\Xi = \left[\underbrace{a, \ldots a}_{p+1}, \xi_{p+1}, \ldots, \xi_n, \underbrace{b, \ldots, b}_{p+1}\right],\ |\Xi| = n + p + 2. \tag{3.9}$$

The space of the B-splines of degree p built over the knot vector Ξ is denoted by

$$\mathcal{S}(\Xi, p) = \text{span}\left\{N_i^p(\xi)\right\}_{i=0}^{n}.$$

Notation 3.9. $\mathcal{S}(\Xi, p)$ will be written with the alternative notation $\mathcal{S}_{\Xi}^{p}$ in case a shorter form is preferable.

Example 3.10. Reconsidering the example 3.3, it is possible to use the same control points to build a B-spline curve. It is necessary to define a knot vector in this case, $\Xi = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$ for instance. The curves in the 2-dimensional and the 3-dimensional space are represented in Figures 3.10 and 3.11.

The properties of the B-spline curves are listed below.

Property1: If $n = p$ and $\Xi = \{a, \ldots, a, b, \ldots, b\}$ then $\boldsymbol{C}(\xi)$ is a Bézier curve.

Property2: *Endpoint interpolation*: $\boldsymbol{C}(a) = \boldsymbol{P}_0$ and $\boldsymbol{C}(b) = \boldsymbol{P}_n$.

Property3: *Variation diminishing* property: no plane has more intersections with the curve than with the control polygon.

Algorithm 3.6 Algorithm for the determination of the derivatives of order $0 \leq k \leq n$ of the i^{th} B-spline basis function in the specified ξ.

```
% derivsBasisFun determines the derivatives of the i-th B-spline basis
% function of degree p built over the knot vector Xi in the point xi of
% degree up to degree n.
% Input:
%   p: degree of the function;
%   Xi: knot vector;
%   i: index of the B-spline basis function to differentiate;
%   xi: point in which to evaliate the derivative;
%   n: max degree of derivative to find;
% Output:
%   derivs: derivs(k) contains the k-th-1 derivative of the function in xi
function derivs = derivsBasisFun(p, Xi, i, xi, n)
% Check if xi is outside the domain.
derivs = zeros(n+1);
if xi < Xi(i+1) || xi >= Xi(i+p+1+1)
    for k = 0:n, derivs(k+1) = 0; end;
    return;
end
for j = 0:p
    if xi >= Xi(i+j+1) && xi < Xi(i+j+2), N(j+1, 1) = 1;
    else N(j+1, 1) = 0; end;
end
% Compute the triangle.
for k = 1:p
    if N(0+1, k-1+1) == 0, saved = 0;
    else saved = ((xi-Xi(i+1)).*N(0+1, k-1+1))./(Xi(i+k+1)-Xi(i+1)); end;
    for j = 0:p-k+1-1
        Xileft = Xi(i+j+1+1);
        Xiright = Xi(i+j+k+1+1);
        if N(j+1+1, k-1+1) == 0
            N(j+1, k+1) = saved;
            saved = 0;
        else
            temp = N(j+1+1, k-1+1)./(Xiright-Xileft);
            N(j+1, k+1) = saved+(Xiright-xi).*temp;
            saved = (xi-Xileft).*temp;
        end
    end
end
for k = 1:n
    for j = 0:k, ND(j+1) = N(j+1, p-k+1); end;
    for jj = 1:k
        if ND(1) == 0, saved = 0;
        else saved = ND(0+1)./(Xi(i+p-k+jj+1)-Xi(i+1)); end;
        for j = 0:k-jj
            Xileft = Xi(i+j+2);
            Xiright = Xi(i+j+p+jj+1);
            if ND(j+1+1) == 0
                ND(j+1) = (p-k+jj).*saved;
                saved = 0;
            else
                temp = ND(j+2)./(Xiright-Xileft);
                ND(j+1) = (p-k+jj).*(saved-temp);
                saved = temp;
            end
        end
    end
    derivs(k+1) = ND(1);
end
```

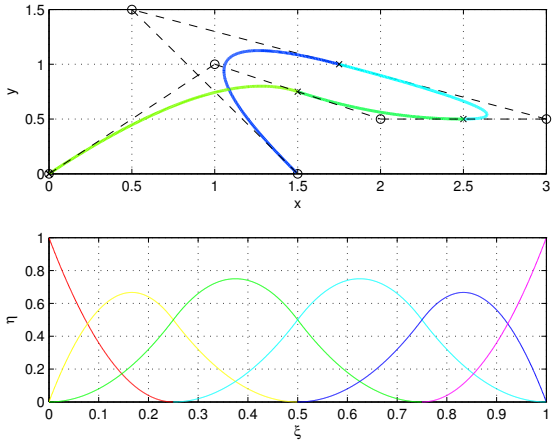

Figure 3.10: Example of B-spline curve in the 2-dimensional space built using the control points of the example 3.3 and the knot vector $\Xi = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. Below the B-spline basis functions used are reported.

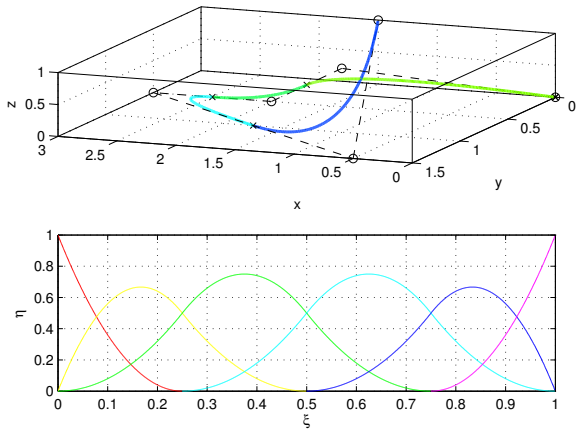

Figure 3.11: Example of B-spline curve in the 3-dimensional space built using the control points of the example 3.3 and the knot vector $\Xi = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. Below the B-spline basis functions used are reported.

Algorithm 3.7 Algorithm to compute the value in the physical space of a curve given its value in the parametric space.

```
% bsplineCurvePoint determines the value of the B-spline curve in the
% point xi of the parametric space.
% Input:
%   n: scalar that indicates that n+1 is the number of control points;
%   p: degree of the cure;
%   Xi: knot vector;
%   P: control points where P(i, j) indicates the j-th coordinate of the
%       i-th control point.
%   xi: point of the parametric space where to evalute the curve.
% Output:
%   C: (xi, eta) is the value of the curve in the two directions.
function C = bsplineCurvePoint(n, p, Xi, P, xi)
% Find the span in which xi lies.
span = findSpan(n, p, xi, Xi);
% Determine all the nonvanishing B-spline basis functions in xi.
N = basisFuns(span, xi, p, Xi);
% Calculation of the value of the curve.
C(2) = 0;
for i = 0:p
    C(1) = C(1)+N(i+1).*P(span-p+i+1, 1);
    C(2) = C(2)+N(i+1).*P(span-p+i+1, 2);
end
```

3.4.5 Algorithm for B-spline curves

Algorithm 3.7 computes the value in the physical space[1] of a curve in a specific point in the parametric space[2].

3.4.6 Algorithm for B-spline curves derivatives

Algorithm 3.8 is an efficient algorithm for evaluating the derivative of a B-spline curve. The base for the algorithm is the equation

$$\frac{\partial^k \boldsymbol{C}(\xi)}{\partial \xi^k} = \sum_{i=0}^{n} \frac{\partial^k N_i^p(\xi)}{\partial \xi} \boldsymbol{P}_i, \tag{3.10}$$

which is a simple application of the linearity of the derivative. Algorithm 3.8 computes all the derivatives of degree $0 \leq k \leq d$ in the vector `CK(k)` and takes advantage of the algorithm presented previously.

3.4.7 Multivariate tensor-product B-splines

We can use the tensor product to obtain B-splines in higher dimension spaces: assume d is the dimension of the B-spline we're trying to build (for $d = 2$ we get B-spline surfaces, for $d = 3$ we get B-spline solids). We need to define d knot vectors denoted with

$$\Xi_\alpha = \left[\underbrace{a_\alpha, \ldots, a_\alpha}_{p_\alpha+1}, \xi_{p_\alpha+1,\alpha}, \ldots, \xi_{n_\alpha,\alpha}, \underbrace{b_\alpha, \ldots, b_\alpha}_{p+1} \right], \ |\Xi_\alpha| = n_\alpha + p_\alpha + 2,$$

[1] The *physical space* will be defined more precisely in 3.4.10.
[2] The *parametric space* will be defined more precisely in 3.4.10.

Algorithm 3.8 Algorithm for the evaluation of the derivative of a B-spline curve.

```
% bsplineCurveDerivs computes the derivatives of a B-spline curve
% up to and including dth derivatives for the curve provided.
% Input:
%   n: defined so that n+1 is the number of control points;
%   p: degree of the B-spline basis function to use;
%   Xi: knot vector in direction xi;
%   P: bidimensional matrix containing the coordinates of the
%      control points on the rows: the kth coordinate of the
%      ith control point is P(i, k);
%   xi: point where to evaluate the derivative of the curve;
%   d: order of the derivative up to which we want to calculate them.
% Output:
%   CK: bidimensional marix containing the derivative: CK(k, i)
%       is the ith coordinate of the kth derivative where 0<=k<=d.
function CK = bsplineCurveDerivs(n, p, Xi, P, xi, d)
dxi = min([d, p]);
CK = zeros(d+1, 3);
% If p < d, higher order derivatives are set to zero.
for k = p+1:d, CK(k+1) = 0; end;
span = findSpan(n, p, xi, Xi);
nderivs = derivsBasisFuns(span, xi, p, dxi, Xi);
for k = 0:dxi
    CK(k+1, length(P(1,:))) = 0;
    for j = 0:p
        CK(k+1, :) = CK(k+1, :)+nderivs(k+1, j+1).*P(span-p+j+1, :);
    end
end
```

with $\alpha = 0, \ldots, d-1$. The tensor product B-spline basis functions can be written as

$$N^{p_0,\ldots,p_d}_{i_0,\ldots,i_d} \triangleq N^{p_0}_{i_0} \otimes \ldots \otimes N^{p_{d-1}}_{i_{d-1}}.$$

The properties of the tensor-product B-spline basis functions for $d = 2$ are listed below.

Property1: This choice of the basis functions guarantees the *local support property*: $N^p_i(\xi) N^q_j(\eta) = 0$ if $(\xi, \eta) \notin [\xi_i, \xi_{i+p+1}) \times [\eta_j, \eta_{j+q+1})$.

Property2: In any given rectangle $[\xi_{i_0}, \xi_{i_0+1}) \times [\eta_{j_0}, \eta_{j_0+1})$ at most $(p+1)(q+1)$ of the $N^p_i(\xi) N^q_j(\eta)$ are nonzero, namely the functions $N^p_i(\xi) N^q_j(\eta)$, for $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$.

Property3: The property of *nonnegativity* states that $N^p_i(\xi) N^q_j(\eta) \geq 0$ for all i, j, p, q, ξ and η.

Property4: The property of *partition of unity* states that $\sum_{i=0}^{n} \sum_{j=0}^{m} N^p_i(\xi) N^q_j(\eta) = 1$ for all $(\xi, \eta) \in [a_0, b_0] \times [a_1, b_1]$.

Property5: $N^p_i(\xi) N^q_j(\eta)$ is $C^\infty((\xi_{i_0}, \xi_{i_0+1}) \times (\eta_{j_0}, \eta_{j_0+1}))$ for all i, j, i_0 and j_0. At a ξ (η) knot it is $p-k$ ($q-k$) times differentiable in the ξ (η) direction, where k is the multiplicity of the knot.

Property6: The set of all bivariate tensor-product B-spline basis functions of p^{th}-degree in the ξ direction and of q^{th}-degree in the η direction $N^p_i(\xi) N^q_j(\eta)$, $i = 0, \ldots, n$ and $j = 0, \ldots, m$ defined on the knot vectors

$$\Xi = \left[\underbrace{\xi_0, \ldots, \xi_0}_{s_{\xi,0}}, \underbrace{\xi_1, \ldots, \xi_1}_{s_{\xi,1}}, \ldots, \underbrace{\xi_k, \ldots, \xi_k}_{s_{\xi,k}} \right]$$

$$H = \left[\underbrace{\eta_0, \ldots, \eta_0}_{s_{\eta,0}}, \underbrace{\eta_1, \ldots, \eta_1}_{s_{\eta,1}}, \ldots, \underbrace{\eta_l, \ldots, \eta_l}_{s_{\eta,l}}\right]$$

forms a *basis of the space* $\mathcal{S}^{p,q}_{\Xi,H}$ *of the of the piecewise-polynomials of degree* p *in the* ξ *direction and degree* q *in the* η *direction, with continuity* $C^{r_{\xi,j}}$ *in direction* ξ *at* ξ_j *with* $r_{\xi,j} = p - s_{\xi,j}$ and with continuity $C^{r_{\eta,j}}$ in direction η at η_j with $r_{\eta,j} = q - s_{\eta,j}$. It is possible to show that the dimension of the space $\mathcal{S}^{p,q}_{\Xi,H}$ is

$$\dim\left(\mathcal{S}^{p,q}_{\Xi,H}\right) = \left(k\,(p+1) - \sum_{j=0}^{k}\left(r_{\xi,j}+1\right)\right)\left(l\,(q+1) - \sum_{j=0}^{l}\left(r_{\eta,j}+1\right)\right). \tag{3.11}$$

The tensor product B-spline space is

$$\begin{aligned}\mathcal{S}\left(\Xi_0, \ldots, \Xi_{d-1}, p_0, \ldots, p_{d-1}\right) &\triangleq \otimes_{\alpha=0}^{d-1}\mathcal{S}\left(\Xi_\alpha, p_\alpha\right), \\ &= \operatorname{span}\left(\left\{N^{p_0,\ldots,p_{d-1}}_{i_0,\ldots,i_{d-1}}\right\}^{n_1,\ldots,n_{d-1}}_{i_0=0,\ldots,i_{d-1}=0}\right).\end{aligned}$$

Notation 3.11. $\mathcal{S}\left(\Xi_0, \ldots, \Xi_{d-1}, p_0, \ldots, p_{d-1}\right)$ will be written with the alternative notation $\mathcal{S}^{p_0,\ldots,p_{d-1}}_{\Xi_0,\ldots,\Xi_{d-1}}$ in case a shorter form is preferable.

Particularly important multivariate B-splines are the B-spline surfaces

$$\boldsymbol{S}\left(\xi,\eta\right) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p\left(\xi\right) N_j^q\left(\eta\right)\boldsymbol{P}_{i,j},\ \boldsymbol{a} \le \boldsymbol{\xi} \le \boldsymbol{b}. \tag{3.12}$$

Example 3.12. B-spline surfaces are built using bivariate tensor-product B-spline basis functions. An example built over the knot vectors $\Xi = H = \{0,0,0,0.5,1,1,1\}$ with $p = q = 2$ can be seen in Figure 3.12.

The $\boldsymbol{P}_{i,j}$'s form a net of control points and basis functions are defined over the knot vectors

$$\Xi = \left[\underbrace{a, \ldots, a}_{p+1}, \xi_{p+1}, \ldots, \xi_n, \underbrace{b, \ldots, b}_{p+1}\right],\ |\Xi| = n + p + 2,$$

$$H = \left[\underbrace{c, \ldots, c}_{q+1}, \eta_{q+1}, \ldots, \eta_m, \underbrace{d, \ldots, d}_{q+1}\right],\ |H| = m + q + 2.$$

Some important properties of the B-spline surfaces are listed below.

Property1: If $n = p$, $m = q$, $\Xi = [a, \ldots, a, b, \ldots, b]$ and $H = [a, \ldots, a, b, \ldots, b]$ then $\boldsymbol{S}\left(\xi,\eta\right)$ is a Bézier surface.

Property2: *Endpoint interpolation*: the surface interpolates the four corner control points $\boldsymbol{S}\left(a_1,a_2\right) = \boldsymbol{P}_{0,0}$, $\boldsymbol{S}\left(b_1,a_2\right) = \boldsymbol{P}_{n,0}$, $\boldsymbol{S}\left(a_1,b_2\right) = \boldsymbol{P}_{0,m}$ and $\boldsymbol{S}\left(b_1,b_2\right) = \boldsymbol{P}_{n,m}$.

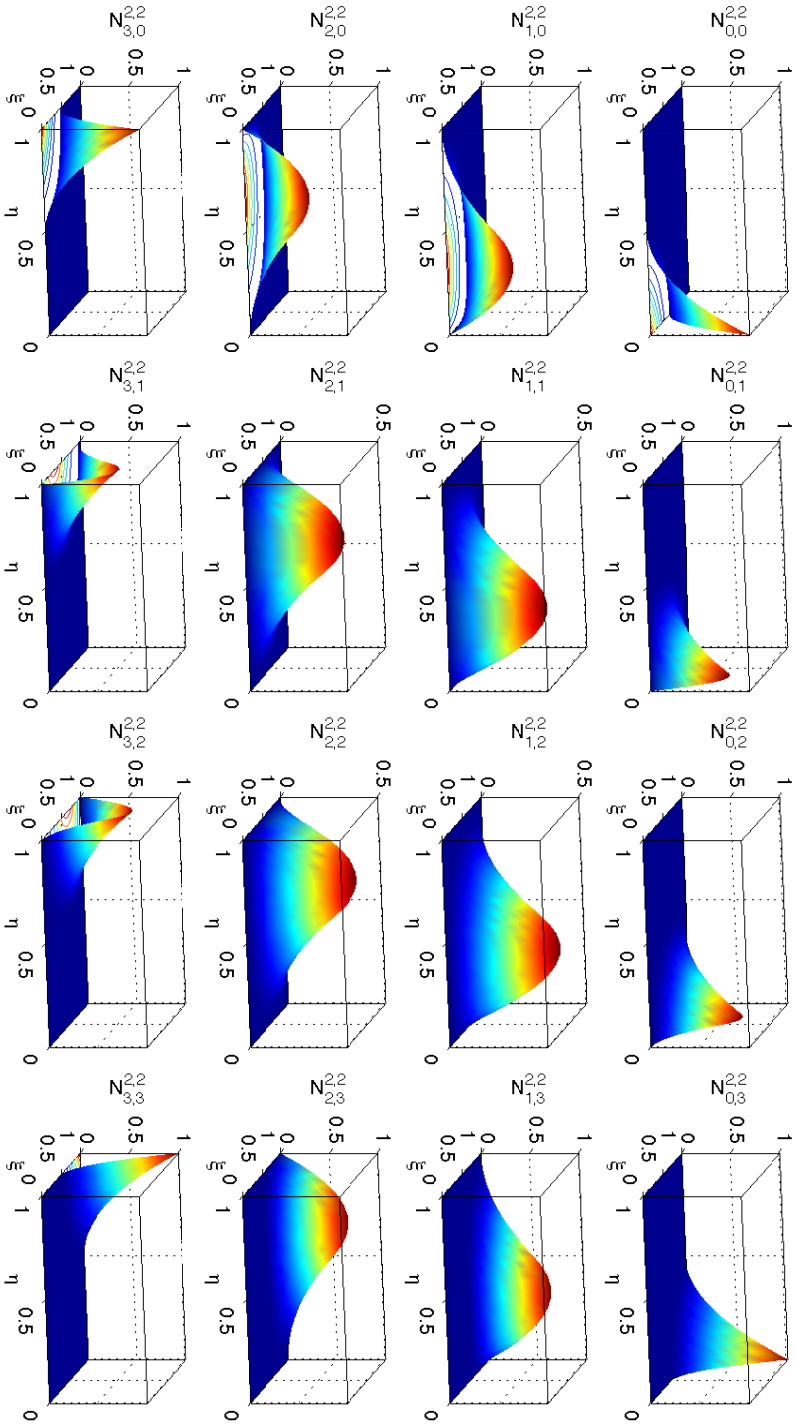

Figure 3.12: Bivariate tensor-product B-spline basis functions built over the knot vectors $\Xi = H = \{0, 0, 0, 0.5, 1, 1, 1\}$ with $p = q = 2$.

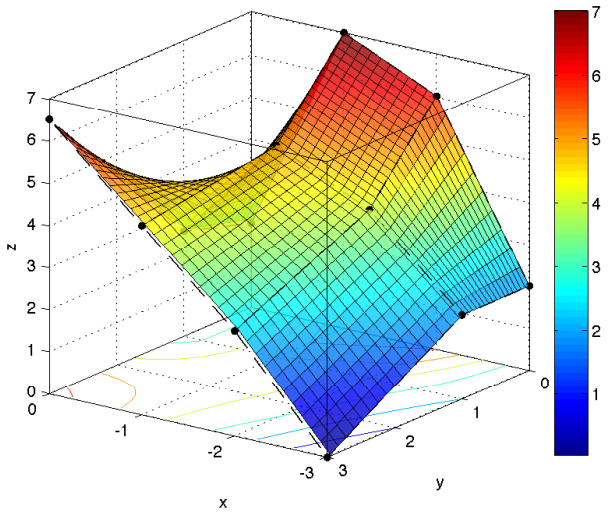

Figure 3.13: Example of B-spline surface built using the same control points of Figure 3.5, $p = q = 1$ and the knot vectors $\Xi = \{0, 0, 0.5, 1, 1\}$ and $H = \{0, 0, 0.\overline{3}, 0.\overline{6}, 1, 1\}$.

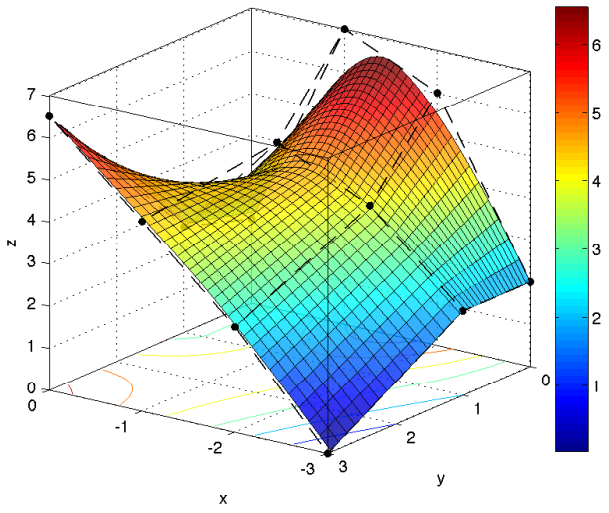

Figure 3.14: Example of B-spline surface built using the same control points of Figure 3.5, $p = 1$, $q = 2$ and the knot vectors $\Xi = \{0, 0, 0.5, 1, 1\}$ and $H = \{0, 0, 0, 0.5, 1, 1, 1\}$.

Example 3.13. Reconsidering the image of Figure 3.5, an example of B-spline surface built using the same control points is shown in Figures 3.13 and 3.14. In the first case, both the degrees p and q are chosen so that the surface interpolates exactly the control points. This is done choosing $p = q = 1$ with the knot vectors $\Xi = \{0, 0, 0.5, 1, 1\}$ and $H = \left\{0, 0, 0.\overline{3}, 0.\overline{6}, 1, 1\right\}$. In Figure 3.14 $q = 2$, so only the other direction exactly interpolates the control points.

Using the tensor product, a solid can be expressed using three sets of basis functions

$$\boldsymbol{S}(\xi, \eta, \zeta) = \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} N_i^p(\xi) N_j^q(\eta) N_k^r(\zeta) \boldsymbol{P}_{i,j,k}, \ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b},$$

where the $\boldsymbol{P}_{i,j,k}$'s are the control points forming the control net, the N_i^p's, the N_j^q'a and the N_k^r's are the B-spline basis functions defined over the knot vectors

$$\Xi = \left[\underbrace{a_1, \ldots, a_1}_{p+1}, \xi_{p+1}, \ldots \xi_n, \underbrace{b_1, \ldots, b_1}_{p+1} \right], \ |\Xi| = n + p + 2,$$

$$H = \left[\underbrace{a_2, \ldots, a_2}_{q+1}, \eta_{q+1}, \ldots \eta_m, \underbrace{b_2, \ldots, b_2}_{q+1} \right], \ |H| = m + q + 2,$$

$$Z = \left[\underbrace{a_3, \ldots, a_3}_{r+1}, \zeta_{r+1}, \ldots \zeta_l, \underbrace{b_3, \ldots, b_3}_{r+1} \right], \ |Z| = l + r + 2.$$

3.4.8 Algorithm for B-spline surfaces

An algorithm for the computation of the value of the B-spline surface in a specified point $[\xi, \eta]^T$ is given in Algorithm 3.9.

3.4.9 Algorithms for B-spline surfaces derivatives

An algorithm for the computation of the value of a B-spline surface derivatives in a specified value of the parametric space comes straightforwardly from the definition in Equation (3.12) with the use of the linearity of the derivative:

$$\frac{\partial^{k+l} \boldsymbol{S}(\xi, \eta)}{\partial^k \xi \partial^l \eta} = \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\partial^k N_i^p(\xi)}{\partial \xi^k} \frac{\partial^l N_j^q(\eta)}{\partial \eta^l} \boldsymbol{P}_{i,j} \tag{3.13}$$

$$= \left[\frac{\partial^k N_i^p(\xi)}{\partial \xi^k} \right]^T [\boldsymbol{P}_{r,s}] \left[\frac{\partial^l N_j^q(\eta)}{\partial \eta^l} \right], \tag{3.14}$$

where, supposing $\xi \in \left[\xi_{\hat{i}}, \xi_{\hat{i}+1}\right)$ and $\eta \in \left[\eta_{\hat{j}}, \eta_{\hat{j}+1}\right)$ we have $\xi_{\hat{i}} - p \leq r \leq \xi_{\hat{i}}$ and $\eta_{\hat{j}} - q \leq s \leq \eta_{\hat{j}}$. An implementation of this is in Algorithm 3.10.

Algorithm 3.9 Algorithm for the computation of the value of a B-spline surface in $[\xi, \eta]^T$.

```
% bsplineSurfPoint evaluates a B-spline surface on a point (xi,eta)
% of the domain.
% Input:
%   n: defined acordingly to the knot vector Xi;
%   p: degree in the first direction;
%   Xi: knot vector in the first direction;
%   m: defined accordingly to the knot vector Eta;
%   q: degree in the second direction;
%   Eta: knot vector in the second direction;
%   P: control points written in the format P(xi, x_k, eta);
%   xi: coordinate in the first direction on which the
%       surface is being evaluated;
%   eta: coordinate in the second direction on which the
%        surface is being evaluated.
% Output:
%   S: value of the surface in the point (xi, eta).
function S = bsplineSurfPoint(n, p, Xi, m, q, Eta, P, xi, eta)
xiSpan = findSpan(n, p, xi, Xi);
etaSpan = findSpan(m, q, eta, Eta);
Nxi = basisFuns(xiSpan, xi, p, Xi);
Neta = basisFuns(etaSpan, eta, q, Eta);
d = length(P(1, 1, :));
S(d) = 0;
for i = 1:d
    S(i) = Nxi*P(xiSpan-p+1:xiSpan+1, etaSpan-q+1:etaSpan+1, i)*Neta';
end
```

3.4.10 Support structures

B-splines and the CAD technologies which will be presented later all share some key concepts and attain compatibility. It is interesting therefore to define some useful structures to work with these technologies, which will be further used in Isogeometric Analysis.

The first concepts are the *parametric space*, which is the space where the domain of the parametric form is defined and the *physical space*, which is the space where the codomain of the parametric form is defined. The splines which will be defined map points in the parametric space in points in the physical space. Another space which can be useful is the *index space*, which is created by plotting the knots equidistantly, regardless of their actual spacing, labeling them with their index. In the parametric space we define, in addiction, the *anchor*, which is the point in the parametric space which lies precisely at the middle of the support of a basis function.

3.5 Univariate and multivariate NURBS's

According to the same principle exposed in the subsection 3.3.1, not all curves and surfaces can be represented with piecewise-polynomials only. Using the same procedure of Subsection 3.3.1 so, we define a B-spline curve where the basis functions are rational functions.

Algorithm 3.10 Algorithm for the computation of the value of the B-spline surface derivatives in a provided point of the parametric space.

```
% Compute B-spline surface derivatives in the specified point                    1
% from order 0 to order d.                                                       2
% Input:                                                                         3
%   n: defined accordingly to the knot vector Xi;                                4
%   p: degree in direction xi;                                                   5
%   Xi: knot vector in direction xi;                                             6
%   m: defined accordingly to the knot vector Eta;                               7
%   q: degree in direction eta;                                                  8
%   Eta: not vector in direction eta;                                            9
%   P: control points written in the format P(xi, eta, x_k);                     10
%   xi: point in direction xi in which the surface is to be determined;          11
%   eta: point in dicrection eta in which the surface is to be determined;       12
%   d: derivatives are to be computed up to order d.                             13
% Output:                                                                        14
%   SKL: tridimensional matrix where SKL_{k,l,i} is the ith                      15
%        coordinate derivative of S(xi, eta) with respect to                     16
%        xi k times and to eta l times.                                          17
%   Nxi: returns derivsBasisFuns(xiSpan, xi, p, d, Xi);                          18
%   Neta: returns derivsBasisFuns(etaSpan, eta, q, d, Eta);                      19
%   spanxi: returns spanxi = findSpan(n, p, xi, Xi);                             20
%   spaneta: returns findSpan(m, q, eta, Eta).                                   21
function [SKL, Nxi, Neta, spanxi, spaneta] =...                                 22
    bsplineSurfDerivs(n, p, Xi, m, q, Eta, P, xi, eta, d)                       23
% Determine the span in which the point [xi, eta]^T is.                         24
spanxi = findSpan(n, p, xi, Xi);                                                25
spaneta = findSpan(m, q, eta, Eta);                                             26
% Determine the derivatives.                                                    27
Nxi = derivsBasisFuns(spanxi, xi, p, d, Xi);                                    28
Neta = derivsBasisFuns(spaneta, eta, q, d, Eta);                                29
SKL = zeros(d+1, length(P(1, 1, :)), d+1);                                      30
for k = 0:d                                                                     31
    for l = 0:d                                                                 32
        for i = 0:length(P(1, 1, :))-1                                          33
            SKL(k+1, i+1, l+1) = Neta(l+1, :)*P(spanxi-p+1:spanxi+1,...         34
                spaneta-q+1:spaneta+1, i+1)'*Nxi(k+1, :)';                      35
        end                                                                     36
    end                                                                         37
end                                                                             38
SKL = permute(SKL, [1, 3, 2]);                                                  39
```

3.5.1 NURBS basis functions

A NURBS basis function can be defined using B-spline basis functions using the relation

$$R_i^p(\xi) = \frac{N_i^p(\xi)\, w_i}{\sum_{\hat{i}=0}^{n} N_{\hat{i}}^p(\xi)\, w_{\hat{i}}},\ a \le \xi \le b, \tag{3.15}$$

where the N_i^p's are the B-spline basis functions already defined in (3.3) over the knot vector

$$\Xi = \left[\underbrace{a,\ldots,a}_{p+1}, \xi_{p+1},\ldots\xi_n, \underbrace{b,\ldots,b}_{p+1}\right],\ |\Xi| = n+p+2,$$

and the w_i's are the weights.

Property1: This choice of the basis functions guarantees the *local support property*: $R_i^p(\xi) = 0$ if $\xi \notin [\xi_i, \xi_{i+p+1})$.

Property2: In any given knot span $[\xi_j, \xi_{j+1})$ at most $p+1$ of the R_i^p are nonzero, namely the functions $R_{j-p}^p, \ldots, R_j^p$.

Property3: The property of *nonnegtivity* states that $R_i^p(\xi) \ge 0$ for all i, p and ξ.

Property4: The property of *partition of unity* states that $\sum_{i=0}^{n} R_j^p(\xi) = 1$ for all $\xi \in [\xi_i, \xi_{i+1})$.

Property5: $R_i^p(\xi)$ is $C^\infty((\xi_j, \xi_{j+1}))$ for all j. At a knot $R_i^p(\xi)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot.

Property6: If $w_i = 1$ for all i then $R_i^p(\xi) = N_i^p(\xi)$. This means that NURBS's are a generalization of the B-splines.

Property7: If no interior knot is defined in the knot vector, then NURBS's are a generalization of Béziers.

Property8: The set of all NURBS basis functions of p^{th}-degree $R_i^p(\xi)$, $i = 0,\ldots,n$ defined on the knot vector

$$\Xi = \left[\underbrace{\xi_0,\ldots,\xi_0}_{s_0}, \underbrace{\xi_1,\ldots,\xi_1}_{s_1}, \ldots, \underbrace{\xi_k,\ldots,\xi_k}_{s_k}\right]$$

forms a *basis of the space* $\mathcal{N}_\Xi^p$ *of the of the piecewise-rational polynomials of degree* p *with continuity* C^{r_j} *at* ξ_j *with* $r_j = p - s_j$. It is possible to show that the dimension of the space $\mathcal{N}_\Xi^p$ is

$$\dim(\mathcal{N}_\Xi^p) = k(p+1) - \sum_{j=0}^{k}(r_j+1).$$

Algorithm 3.11 Algorithm for the evaluation of the i^{th} NURBS basis function.

```
% NURBSBasisFun computes the derivatives of the i-th NURBS basis function.
% Input:
%   i: index of the NURBS basis function;
%   xi: point where to evaluate the derivative;
%   n: number of control points minus 1;
%   p: degree of the NURBS basis function;
%   Xi: knot vector over which the NURBS basis function has to be built;
%   w: vector of the weights.
% Output:
%   R: value of the derivative.
function R = NURBSBasisFun(i, xi, n, p, Xi, w)
spanXi = findSpan(n, p, xi, Xi);
if i < spanXi-p || i > spanXi, R = 0;
else
    N = basisFuns(spanXi, xi, p, Xi);
    R = N(p+1-(spanXi-i)).*w(i+1)./(N*w(spanXi-p+1:spanXi+1)');
end
```

3.5.2 Algorithm for NURBS basis functions

It is important to have an efficient way of evaluating both a NURBS basis function and its derivative: Algorithm 3.11 and Algorithm 3.12 respectively can be used.

In Algorithm 3.11, we simply need to compute

$$R_i^p(\xi) = \frac{N_i^p(\xi)\, w_i}{\sum_{\hat{i}=0}^{n} N_{\hat{i}}^p(\xi)\, w_{\hat{i}}};$$

the terms $N_i^p(\xi)$'s can be computed with a single call to `basisFuns(...)`.

3.5.3 Algorithm for NURBS basis functions derivatives

Algorithm 3.12 computes the function $\partial R_i^p / \partial \xi$ in ξ:

$$\frac{\partial R_i^p(\xi)}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\frac{N_i^p(\xi)\, w_i}{\sum_{\hat{i}=0}^{n} N_{\hat{i}}^p(\xi)\, w_{\hat{i}}}\right).$$

By the linearity of the derivative, we can say

$$\frac{\partial R_i^p(\xi)}{\partial \xi} = \frac{w_i \frac{\partial N_i^p(\xi)}{\partial \xi} \cdot W(\xi) - N_i^p(\xi)\, w_i \cdot \sum_{\hat{i}=0}^{n} \frac{\partial N_{\hat{i}}^p(\xi)}{\partial \xi} w_{\hat{i}}}{\left(W(\xi)\right)^2},$$

where

$$W(\xi) = \sum_{\hat{i}=0}^{n} N_{\hat{i}}^p(\xi)\, w_{\hat{i}}.$$

The N_i^p's and the $\partial N_i^p / \partial \xi$'s can all be computed with a single call to, respectively, `basisFuns(...)` and `derivsBasisFuns(...)` (according to our definition of

Algorithm 3.12 Algorithm for the evaluation of the derivative of a NURBS basis function.

```
% derivsNURBSBasisFun computes the value of the derivative of
% the i-th NURBS basis function in the point xi.
% Input:
%   i: index of the NURBS basis function to compute;
%   xi: point in which to compute the NURBS basis function;
%   p: degree of the NURBS basis function;
%   Xi: knot vector over which the NURBS basis function have to be
%       computed.
%   w: vector containing the weights.
% Output:
%   R: value of the i-th NURBS basis function computed in xi.
function R = derivsNURBSBasisFuns(i, xi, p, Xi, w)
n = length(Xi)-p-2;
spanXi = findSpan(n, p, xi, Xi);
% Computation of all the nonvanishing B-spline basis functions in xi.
Nips = basisFuns(spanXi, xi, p, Xi);
% Computation of the value for the denominator of the NURBS basis function
W = Nips*w(spanXi-p+1:spanXi+1)';
% Compitatio of the derivaives of the B-spline basis functions.
dNips = derivsBasisFuns(spanXi, xi, p, 1, Xi);
% Linear combinatio of the derivatives.
Cw = w(spanXi-p+1:spanXi+1)*dNips(2, :)';
% Values of the i-th basis function and relative derivative.
Nip = basisFun(p, n+p+1, Xi, i, xi);
dNip = derivsBasisFun(p, Xi, i, xi, 1);
% Final result.
R = (w(i+1).*dNip(1+1).*W - Nip.*w(i+1).*Cw)./(W.^2);
```

`derivsBasisFuns(...)`, a simple call to this function would be sufficient, as it computes the derivatives of degree $k = 0$ as well).

Example 3.14. An example of derivatives of NURBS basis functions can be seen in Figure 3.15. Cubic NURBS basis functions are plotted over the knot vector $\Xi = \{0, 0, 0, 0, 1, 4, 6, 8, 8, 8, 8\}$ with the weights $\boldsymbol{w} = [1, 1, 1, 3, 1, 1, 1]$. In Figure 3.15b the respective derivatives are represented.

3.5.4 Univariate NURBS

As a result, a p^{th}-degree NURBS (Non Uniform Rational B-spline) curve (or univariate NURBS) is

$$\boldsymbol{C}(\xi) = \frac{\sum_{i=0}^{n} N_i^p(\xi)\, w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} N_i^p(\xi)\, w_i}, \; a \leq \xi \leq b, \tag{3.16}$$

where the $\boldsymbol{P}_i$'s are the *control points* forming the *control polygon*, which (3.16) can be rewritten

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} R_i^p(\xi)\, \boldsymbol{P}_i, \; a \leq \xi \leq b, \tag{3.17}$$

It is possible moreover to use homogeneous coordinates to get a better representation of the NURBS:

$$\boldsymbol{C}^w(\xi) = \sum_{i=0}^{n} N_i^p(\xi)\, \boldsymbol{P}_i^{w_i}, \; a \leq \xi \leq b. \tag{3.18}$$

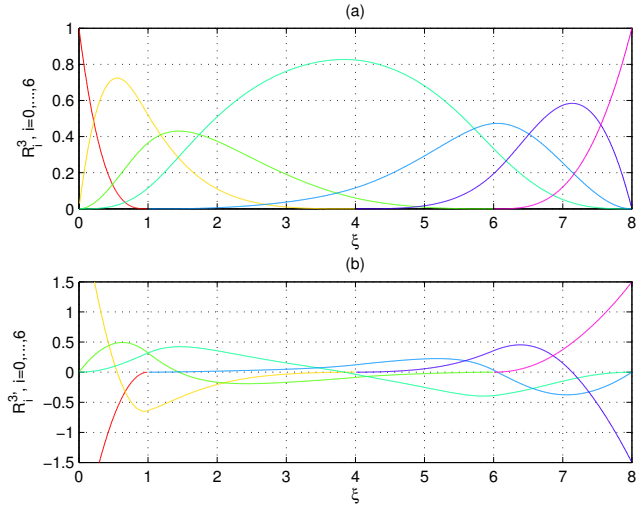

Figure 3.15: Representation of cubic NURBS basis functions are plotted over the knot vector $\Xi = \{0, 0, 0, 0, 1, 4, 6, 8, 8, 8, 8\}$ with the weights $\boldsymbol{w} = [1, 1, 1, 3, 1, 1, 1]$ in (a) and the respective derivatives in (b).

The properties of the NURBS curves are listed below.

Property1: If $n = p$ and $\Xi = \{a, \ldots, a, b, \ldots, b\}$ then $\boldsymbol{C}(\xi)$ is a Bézier curve.

Property2: *Endpoint interpolation*: $\boldsymbol{C}(a) = \boldsymbol{P}_0$ and $\boldsymbol{C}(b) = \boldsymbol{P}_n$.

Property3: *Variation diminishing* property: no plane has more intersections with the curve than with the control polygon.

Example 3.15. In Figure 3.16 a circle has been built using a NURBS curve. The NURBS is drawn using the data reported in Table 3.3.

i	$\boldsymbol{P}_i$	w_i
0	$[1, 0]^{\mathrm{T}}$	1
1	$[1, 1]^{\mathrm{T}}$	$1/\sqrt{2}$
2	$[0, 1]^{\mathrm{T}}$	1
3	$[-1, 1]^{\mathrm{T}}$	$1/\sqrt{2}$
4	$[-1, 0]^{\mathrm{T}}$	1
5	$[-1, -1]^{\mathrm{T}}$	$1/\sqrt{2}$
6	$[0, -1]^{\mathrm{T}}$	1
7	$[1, -1]^{\mathrm{T}}$	$1/\sqrt{2}$
8	$[1, 0]^{\mathrm{T}}$	1

Table 3.3: Data used for the construction of the NURBS circle of Figure 3.16.

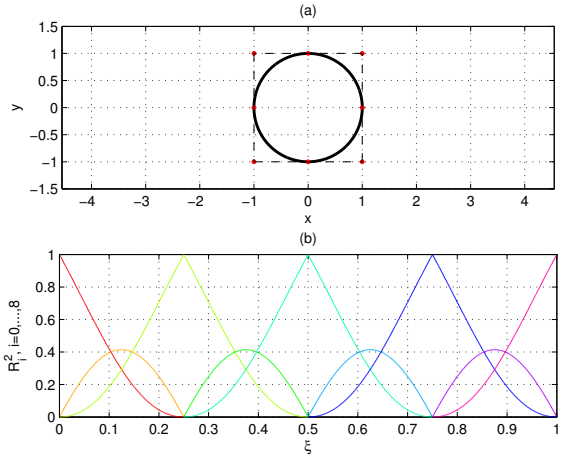

Figure 3.16: Representation of a unit circle with its control points in (a) and of the NURBS basis functions used to build it in (b).

3.5.5 Algorithm for NURBS curves

Algorithms 3.13 and 3.14 can be used to evaluate a NURBS curve and its derivative. Algorithm 3.13 is a straightforward application of the definition of univariate NURBS of Equation (3.16).

3.5.6 Algorithm for NURBS curves derivatives

It is possible to prove (see [24]) that the k^{th} derivative of a NURBS curve $\boldsymbol{C}^{(k)}(\xi)$ can be computed using the equation

$$\boldsymbol{C}^{(k)}(\xi) = \frac{\boldsymbol{A}^{(k)}(\xi) - \sum_{i=1}^{k} \binom{k}{i} w^{(i)}(\xi)\, \boldsymbol{C}^{(k-i)}(\xi)}{w(\xi)},$$

where $\boldsymbol{A}^{(k)}(\xi)$ is the vector-valued function which contains the first three components of the k^{th} derivative of $\boldsymbol{C}^{w}(\xi)$ and $w^{(i)}(\xi)$ is the i^{th} derivative of the fourth component of $\boldsymbol{C}^{w}(\xi)$ (both can be computed with Equation (3.10) and Algorithm 3.8) .

3.5.7 Multivariate tensor-product NURBS

Similarly, the form of a NURBS surface is

$$\boldsymbol{S}(\xi,\eta) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(\xi)\, N_j^q(\eta)\, w_{i,j}\boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(\xi)\, N_j^q(\eta)\, w_{i,j}}, \quad \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}$$

Algorithm 3.13 Algorithm for the evaluation of a NURBS curve.

```
% NURBSCurvePoint evaluates a NURBS curve in the specified
% point xi.
% Input:
%   n: defined accordingly to the knot vector;
%   p: degree of the NURBS curve;
%   Xi: knot vector on which the NURBS is defined;
%   Pw: weighted control points written in rows;
%   xi: point in which the vurve has to be evaluated.
% Output:
%   C: value of the curve in the point xi.
function C = NURBSCurvePoint(n, p, Xi, Pw, xi)
span = findSpan(n, p, xi, Xi);
N = basisFuns(span, xi, p, Xi);
d = length(Pw(1, :));
Cw = zeros(1, d);
for j = 0:p
    Cw(1:d) = Cw(1:d) + N(j+1).*Pw(span-p+j+1, 1:d);
end
C(1:d) = Cw(1:d)./Cw(d);
```

Algorithm 3.14 Algorithm for the evaluation of the derivatives of a NURBS curve.

```
% NURBSCurveDerivs computes all the k derivatives 0 <= k <= d of a NURBS
% curve.
% Input:
%   n: number of the control points minus one;
%   p: degree of the curve;
%   Xi: knot vector over which the curve has to be built;
%   Pw: weighted control points;
%   xi: point where to evaluate the derivative;
%   d: maximum degree of the derivative to calculate.
% Output:
%   CK: CK(k) is the k-th derivative of the curve for 0 <= k <= d.
function CK = NURBSCurveDerivs(n, p, Xi, Pw, xi, d)
Aders = bsplineCurveDerivs(n, p, Xi, Pw, xi, d);
wders = Aders(:, 3);
Aders = Aders(:, 1:2);
CK(1, :) = Aders(1, 1:2);
for k = 0:d
    v = Aders(k+1, :);
    for i = 1:k
        v = v-nchoosek(k, i).*wders(i+1).*CK(k-i+1, :);
    end
    CK(k+1, :) = v./wders(0+1);
end
```

where the $\boldsymbol{P}_{i,j}$'s are the control points forming the control net, the N_i^p's and the N_j^q's are the B-spline basis functions already defined in (3.3) over the knot vectors

$$\Xi = \left\{ \underbrace{a_1, \ldots, a_1}_{p+1}, \xi_{p+1}, \ldots \xi_n, \underbrace{b_1, \ldots, b_1}_{p+1} \right\}, \ |\Xi| = n + p + 2,$$

$$H = \left\{ \underbrace{a_2, \ldots, a_2}_{q+1}, \eta_{q+1}, \ldots \eta_m, \underbrace{b_2, \ldots, b_2}_{q+1} \right\}, \ |H| = m + q + 2,$$

where the $w_{i,j}$'s are the *weights*. As already done, it is possible to use the rational basis functions

$$R_{i,j}^{p,q}(\xi, \eta) = \frac{N_i^p(\xi) N_j^q(\eta) w_{i,j}}{\sum_{\hat{i}=0}^{n} \sum_{\hat{j}=0}^{m} N_{\hat{i}}^p(\xi) N_{\hat{j}}^q(\eta) w_{\hat{i},\hat{j}}}, \ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}, \tag{3.19}$$

to give another definition of a NURBS surface

$$\boldsymbol{S}(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}^{p,q}(\xi, \eta) \boldsymbol{P}_{i,j}, \ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}.$$

Again, homogeneous coordinates con be employed to give a more manageable description of a NURBS surface:

$$\boldsymbol{S}^w(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_i^p(\xi) N_j^q(\eta) \boldsymbol{P}_{i,j}^{w_{i,j}}, \ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}. \tag{3.20}$$

The following are the properties of the bivariate tensor-product NURBS basis functions.

Property1: This choice of the basis functions guarantees the *local support property*: $R_{i,j}^{p,q}(\xi, \eta) = 0$ if $(\xi, \eta) \notin [\xi_i, \xi_{i+p+1}) \times [\eta_j, \eta_{j+q+1})$.

Property2: In any given rectangle $[\xi_{i_0}, \xi_{i_0+1}) \times [\eta_{j_0}, \eta_{j_0+1})$ at most $(p+1)(q+1)$ of the $R_{i,j}^{p,q}(\xi, \eta)$ are nonzero, namely the functions $R_{i,j}^{p,q}(\xi, \eta)$, for $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$.

Property3: The property of *nonnegativity* states that $R_{i,j}^{p,q}(\xi, \eta) \geq 0$ for all i, j, p, q, ξ and η.

Property4: The property of *partition of unity* states that $\sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}^{p,q}(\xi, \eta) = 1$ for all $(\xi, \eta) \in [a_0, b_0] \times [a_1, b_1]$.

Property5: $R_{i,j}^{p,q}(\xi, \eta)$ is $C^{\infty}((\xi_{i_0}, \xi_{i_0+1}) \times (\eta_{j_0}, \eta_{j_0+1}))$ for all i, j, i_0 and j_0. At a ξ (η) knot it is $p-k$ ($q-k$) times differentiable in the ξ (η) direction, where k is the multiplicity of the knot.

Property6: If all $w_{i,j} = a$ and $a \neq 0$ then $R_{i,j}^{p,q}(\xi, \eta) = N_{i,j}^{p,q}(\xi, \eta)$.

i	$\boldsymbol{P}_{i,1}$	$\boldsymbol{P}_{i,2}$	$\boldsymbol{P}_{i,3}$	$w_{i,1}$	$w_{i,2}$	$w_{i,3}$
1	$[-1,0]^T$	$[-2.5,0]^T$	$[-4,0]^T$	1	1	1
2	$\left[-1,\sqrt{2}-1\right]^T$	$[-2.5,0.75]^T$	$[-4,4]^T$	$(1+1/\sqrt{2})/2$	1	1
3	$\left[0-\sqrt{2},1\right]^T$	$[-0.75,2.5]^T$	$[-4,4]^T$	$(1+1/\sqrt{2})/2$	1	1
4	$[0,1]^T$	$[0,2.5]^T$	$[0,4]^T$	1	1	1

Table 3.4: Data used for the construction of the NURBS surface of Figure 3.18.

Property7: The set of all bivariate tensor-product B-spline basis functions of p^{th}-degree in the ξ direction and of q^{th}-degree in the η direction $R_{i,j}^{p,q}(\xi,\eta)$, $i=0,\ldots,n$ and $j=0,\ldots,m$ defined on the knot vectors

$$\Xi = \left[\underbrace{\xi_0,\ldots,\xi_0}_{s_{\xi,0}},\underbrace{\xi_1,\ldots,\xi_1}_{s_{\xi,1}},\ldots,\underbrace{\xi_k,\ldots,\xi_k}_{s_{\xi,k}}\right]$$

$$H = \left[\underbrace{\eta_0,\ldots,\eta_0}_{s_{\eta,0}},\underbrace{\eta_1,\ldots,\eta_1}_{s_{\eta,1}},\ldots,\underbrace{\eta_l,\ldots,\eta_l}_{s_{\eta,l}}\right]$$

forms a *basis of the space* $\mathcal{N}_{\Xi,H}^{p,q}$ *of the piecewise-rational polynomials of degree* p *in the* ξ *direction and degree* q *in the* η *direction, with continuity* $C^{r_{\xi,j}}$ *in direction* ξ *at* ξ_j *with* $r_{\xi,j}=p-s_{\xi,j}$ and with continuity $C^{r_{\eta,j}}$ in direction η at η_j with $r_{\eta,j}=q-s_{\eta,j}$. It is possible to show that the dimension of the space $\mathcal{N}_{\Xi,H}^{p,q}$ is

$$\dim\left(\mathcal{N}_{\Xi,H}^{p,q}\right)=\left(k\,(p+1)-\sum_{j=0}^{k}(r_{\xi,j}+1)\right)\left(l\,(q+1)-\sum_{j=0}^{l}(r_{\eta,j}+1)\right).$$

Example 3.16. Figure 3.17 shows the bivariate tensor-product NURBS basis functions built over the knot vectors $\Xi=H=\{0,0,0,0.5,1,1,1\}$ with $p=q=2$.

Example 3.17. Figure 3.18 shows a surface drawn with a NURBS surface. The hole drawn in corner is quarter of a cirle and it requires a NURBS to be drawn. The data necessary to draw the surface is reported in Table 3.4.

Some important properties of the B-spline surfaces are listed below.

Property1: If $n=p$, $m=q$, $\Xi=[a,\ldots,a,b,\ldots,b]$ and $H=[a,\ldots,a,b,\ldots,b]$ then $\boldsymbol{S}(\xi,\eta)$ is a Bézier surface.

Property2: *Endpoint interpolation*: the surface interpolates the four corner control points $\boldsymbol{S}(a_1,a_2)=\boldsymbol{P}_{0,0}$, $\boldsymbol{S}(b_1,a_2)=\boldsymbol{P}_{n,0}$, $\boldsymbol{S}(a_1,b_2)=\boldsymbol{P}_{0,m}$ and $\boldsymbol{S}(b_1,b_2)=\boldsymbol{P}_{n,m}$.

Using the tensor product, a solid can be expressed using three sets of basis

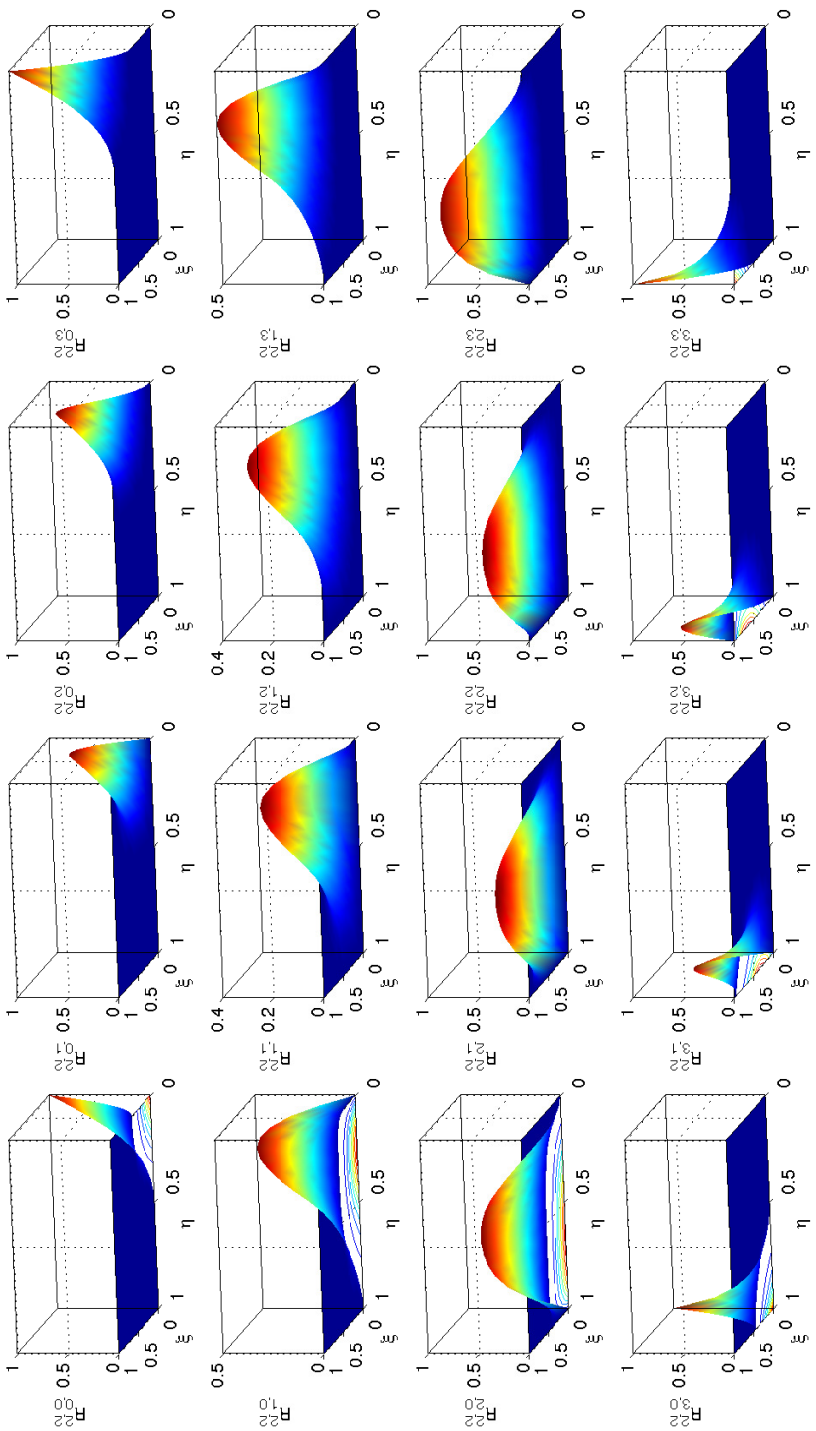

Figure 3.17: Bivariate tensor-product NURBS basis functions built over the knot vectors $\Xi = H = \{0, 0, 0, 0.5, 1, 1, 1\}$ with $p = q = 2$.

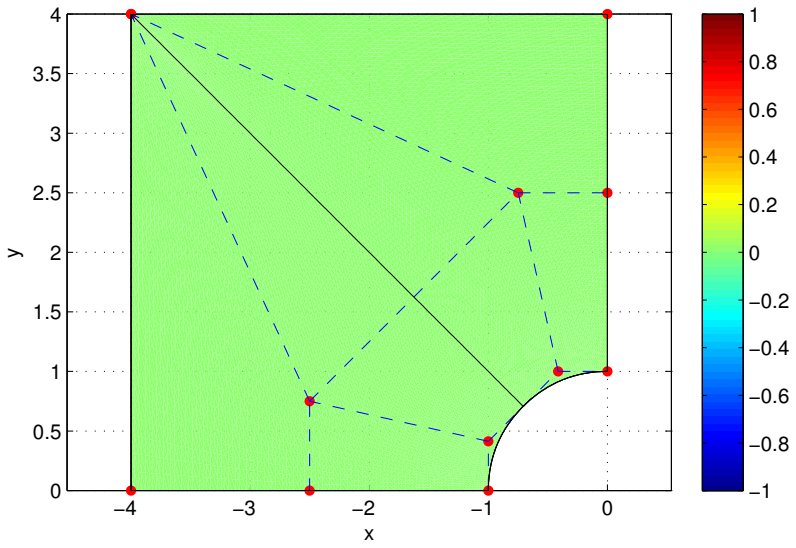

Figure 3.18: Representation of a square surface with a hole in a corner.

functions

$$\boldsymbol{S}(\xi,\eta,\zeta) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} N_i^p(\xi)\, N_j^q(\eta)\, N_k^r(\zeta)\, w_{i,j,k}\boldsymbol{P}_{i,j,k}}{\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} N_i^p(\xi)\, N_j^q(\eta)\, N_k^r(\zeta)\, w_{i,j,k}}, \quad \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}. \tag{3.21}$$

where the $\boldsymbol{P}_{i,j,k}$'s are the control points of the solid, the N_i^p's, the N_j^q's and the N_k^r's are the B-spline basis functions defined in (3.3) over the knot vectors

$$\Xi = \left[\underbrace{a_1,\ldots,a_1}_{p+1},\xi_{p+1},\ldots\xi_n,\underbrace{b_1,\ldots,b_1}_{p+1}\right],\ |\Xi| = n+p+2,$$

$$H = \left[\underbrace{a_2,\ldots,a_2}_{q+1},\eta_{q+1},\ldots\eta_m,\underbrace{b_2,\ldots,b_2}_{q+1}\right],\ |H| = m+q+2,$$

$$Z = \left[\underbrace{a_3,\ldots,a_3}_{r+1},\zeta_{r+1},\ldots\zeta_l,\underbrace{b_3,\ldots,b_3}_{r+1}\right],\ |Z| = l+r+2.$$

Algorithm 3.15 Algorithm for evaluating the value of a NURBS surface in (ξ, η).

```
% NURBSSurfPoint evaluates a NURBS surface on a point (xi, eta)
% of the domain.
% Input:
%   n: defined acordingly to the knot vector Xi;
%   p: degree in the first direction;
%   Xi: knot vector in the first direction;
%   m: defined accordingly to the knot vector Eta;
%   q: degree in the second direction;
%   Eta: knot vector in the second direction;
%   Pw: weighted points written in the format P(xi, x_k, eta);
%   xi: coordinate in the first direction on which the
%       surface is being evaluated;
%   eta: coordinate in the second direction on which the
%        surface is being evaluated.
% Output:
%   S: value of the surface in the point (xi, eta).
function [S] = NURBSSurfPoint(n, p, Xi, m, q, Eta, Pw, xi, eta)
xiSpan = findSpan(n, p, xi, Xi);
etaSpan = findSpan(m, q, eta, Eta);
Nxi = basisFuns(xiSpan, xi, p, Xi);
Neta = basisFuns(etaSpan, eta, q, Eta);
d = length(Pw(1, 1, :));
Sw(d) = 0;
for i = 1:d
    Sw(i) = Nxi*Pw(xiSpan-p+1:xiSpan+1, etaSpan-q+1:etaSpan+1, i)*Neta';
end
S = Sw(1:d-1)./Sw(d);
```

and the $w_{i,j,k}$'s are the weights. Defining the B-spline rational basis functions

$$R^{p,q,l}_{i,j,k}(\xi,\eta,\zeta) = \frac{N^p_i(\xi)\, N^q_j(\eta)\, N^l_k(\zeta)\, w_{i,j,k}}{\sum_{\hat{i}=0}^{n}\sum_{\hat{j}=0}^{m}\sum_{\hat{k}=0}^{l} N^p_{\hat{i}}(\xi)\, N^q_{\hat{j}}(\eta)\, N^l_{\hat{k}}(\zeta)\, w_{\hat{i},\hat{j},\hat{k}}},\ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b},$$

(3.21) can be written in a simpler form:

$$\boldsymbol{S}(\xi,\eta,\zeta) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} R^{p,q,l}_{i,j,k}(\xi,\eta,\zeta)\, \boldsymbol{P}_{i,j,k},\ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}.$$

As usual, the usage of homogeneous coordinates reduces (3.21) to

$$\boldsymbol{S}^w(\xi,\eta,\zeta) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} N^p_i(\xi)\, N^q_j(\eta)\, N^l_k(\zeta)\, \boldsymbol{P}^{w_{i,j,k}}_{i,j,k},\ \boldsymbol{a} \leq \boldsymbol{\xi} \leq \boldsymbol{b}.$$

3.5.8 Algorithm for NURBS surfaces

Equation (3.20) is used in Algorithm 3.15 to determine efficiently the value of the surface in (ξ, η).

3.5.9 Algorithm for NURBS surfaces derivatives

It is possible to use Algorithm 3.16 to evaluate the derivative of a NURBS surface. It uses Algorithm 3.10 and Equations (3.14) to evaluate the derivatives

Algorithm 3.16 Algorithm for the evaluation of the derivative of a NURBS surface.

```
% NURBSSurfDerivs evaluates the derivatives of the NURBS surface S(xi, eta)
% of order up to 0<=k+l<=d, k times with respect to xi and l times with
% respect to eta.
% Input:
%   n: defined accordingly to the knot vector Xi;
%   p: degree in direction xi;
%   Xi: knot vector in direction Xi;
%   m: defined accordingl to the knot vector Eta;
%   q: degree in direction eta;
%   Eta: knot vector in direction eta;
%   Pw: weighted control points;
%   xi: value in which to evaluate the surface in the xi direction;
%   eta: value in which to evaluate the surface in the eta direction;
% Output:
%   SKL: derivatives of the NURBS surface S(xi, eta)
%        of order up to 0<=k+l<=d, k times with respect to xi and l times
%        with respect to eta. SKL(k, l) contains the derivatives of the
%        surface differentiated k times with respect to xi and l times with
%        respect to eta.
function [SKL] = NURBSSurfDerivs(n, p, Xi, m, q, Eta, Pw, xi, eta, d)
Aders = bsplineSurfDerivs(n, p, Xi, m, q, Eta, Pw, xi, eta, d);
wders = Aders(:, :, end);
Aders = Aders(:, :, 1:end-1);
Bders = permute(Aders, [1, 3, 2]);
SKL = zeros(d+1, d+1);
for k = 0:d
    for l = 0:d-k
        v = Bders(k+1, :, l+1);
        for j = 1:l
            v = v-nchoosek(l, j).*wders(0+1, j+1).*SKL(k+1, :, l-j+1);
        end
        for i = 1:k
            v = v-nchoosek(k, i).*wders(i+1, 0+1).*SKL(k-i+1, :, l+1);
            v2 = 0;
            for j = 1:l
                v2 = v2+nchoosek(l, j).*wders(i+1, j+1).*...
                    SKL(k-i+1, :, l-j+1);
            end
            v = v-nchoosek(k, i).*v2;
        end
        SKL(k+1, :, l+1) = v./wders(0+1, 0+1);
    end
end
SKL = permute(SKL, [1, 3, 2]);
```

of $\boldsymbol{S}^w(\xi,\eta)$ ($\boldsymbol{S}^w(\xi,\eta)$ can, as said, be expressed by a B-spline curve where instead of points with three components, four components weighted points are used). Starting from $\boldsymbol{S}^w(\xi,\eta)$ we can compute the derivatives of the NURBS surface $\boldsymbol{S}(\xi,\eta)$ with the equation

$$\frac{\partial^{k+l}\boldsymbol{S}(\xi,\eta)}{\partial\xi^k\partial\eta^l} = \frac{1}{w}\left(\frac{\partial^{k+l}\boldsymbol{A}(\xi,\eta)}{\partial\xi^k\partial\eta^l} - \sum_{i=1}^{k}\binom{k}{i}\frac{\partial^i w(\xi,\eta)}{\partial\xi^i}\frac{\partial^{k-i+l}\boldsymbol{S}}{\partial\xi^{k-i}\partial\eta^l} + \right.$$

$$\left. - \sum_{j=1}^{l}\binom{l}{j}\frac{\partial^j w}{\partial\eta^j}\frac{\partial^{k+l-j}\boldsymbol{S}(\xi,\eta)}{\partial\xi^k\partial\eta^{l-j}} - \sum_{i=1}^{k}\binom{k}{i}\sum_{j=1}^{l}\binom{l}{j}\frac{\partial^{i+j}w(\xi,\eta)}{\partial\xi^i\partial\eta^j}\frac{\partial^{k-i+l-j}\boldsymbol{S}(\xi,\eta)}{\partial\xi^{k-i}\partial\eta^{l-j}}\right).$$

3.6 T-splines

An interesting technology from the point of view of both CAD and Isogeometric Analysis is the T-spline technology, initially introduced by Sederberg in [28]. T-splines allows:

1. add details only where necessary;
2. maintain NURBS compatibility;
3. allow watertight placement of patches;
4. reduce the total number of control points.

3.6.1 T-mesh and basis functions

The definition of a T-spline begins from the index space of a *T-mesh*, where knots are defined as before, and are placed equidistantly on a rectangle. In this case, however, *T-junctions* are allowed: T-junctions are vertices formed by the intersection of three edges. This is not possible in NURBS and B-splines.

Let's consider a B-spline basis function N_α^p: according to the properties already explained, it's support comprise only the knots ξ_α^i to $\xi_{\alpha+p+1}^i$. These knots form the local knot vector Ξ_α^i, where i is the direction, and the corresponding anchor is $\boldsymbol{s}_\alpha$.

If p is odd, for each index space direction, we create a local knot vector Ξ_α^1 and Ξ_α^2, which are empty at the beginning. Next, we place ξ_i^1 in Ξ_α^1 and ξ_j^2 in Ξ_α^2 supposing $\boldsymbol{s}_\alpha = \{i, j\}$. Next, we travel horizontally on one side of the anchor, and for each orthogonal edge k encountered, we add ξ_k^1 in Ξ_α^1 until we have added a total of $^{p+1}/_2$ knots. The same has to be done travelling horizontally on the other side of the anchor until we've added a total of $^{p+1}/_2$ knots. On both sides, if less than $^{p+1}/_2$ knots can be added as no more orthogonal edges are found, we repeat the last one until that value is reached. The same process is repeated in the other direction.

In case p is even, anchors are located inside an element and not at intersections. So, the same process is repeated, without adding the first knot to the knot vector and we insert a total of $^p/_2 + 1$ knots instead of $^{p+1}/_2$.

For a given T-mesh and degree p, let $A \subset \mathbb{Z}^2$ be the index set containing every α for which $\boldsymbol{s}_\alpha$ is an anchor. Using the Ξ_α^1's and the Ξ_α^2's we are able to define B-spline basis functions $N_\alpha^p(\boldsymbol{\xi})$ defined in the parametric space. For each $\alpha \in A$ we define a control point $\boldsymbol{P}_\alpha \in \mathbb{R}^d$ (d can be chosen accordingly to the dimension of the physical space) and a weight w_α to construct a set of *T-spline blending functions*

$$R_\alpha(\xi) = \frac{w_\alpha N_\alpha^p(\boldsymbol{\xi})}{\sum_{\beta \in A} w_\beta N_\beta^p(\boldsymbol{\xi})},$$

for which the *partition of unity* is still valid.

The same definitions and procedures can be generalized almost straightforwardly to three-dimensional spaces to create T-spline solids.

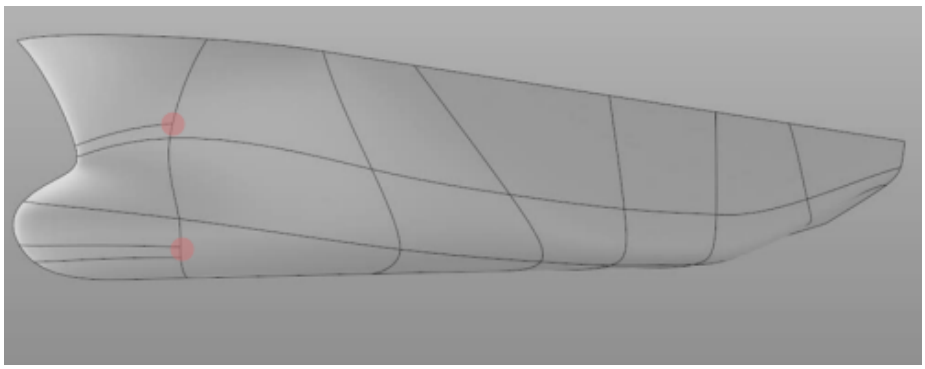

Figure 3.19: Partial isoparms and T-points.

3.6.2 Advantages

T-Splines surfaces can contain areas with differing levels of detail (see Figure 3.19). Control points may be added only where needed so that a typical T-Spline surfaces will have up to 50% fewer control points than the identical equivalent set of NURBS surfaces. In other words, T-Splines are similar to NURBS, with the difference that you can have partial *isoparms* (isocurves or isosurfaces). A main difference between T-Splines and NURBS is the existence of *T-points*: vertices where on one side, there is an isoparm, and on the other side, there isn't. The surface is always smooth (C^2) at a T-point. NURBS's don't allow T-points.

There are various types of surfaces for which it is necessary to use multiple NURBS surfaces (polysurfaces) to be drawn: extrusions, holes, and other unique features are easy to create in a T-Spline surface. This is another difference between T-Splines and NURBS. NURBS require multiple surfaces, or a polysurface, for such objects. T-Splines can accommodate these features in a single surface, by using a special point called a *star point* (see Figure 3.20).

T-splines are a generalization of NURBS's, so there is a complete backcompatibility (see Figure 3.21 and 3.22), which is an important feature when T-splines are to be introduced in a industrial environment, where most models are NURBS-based.

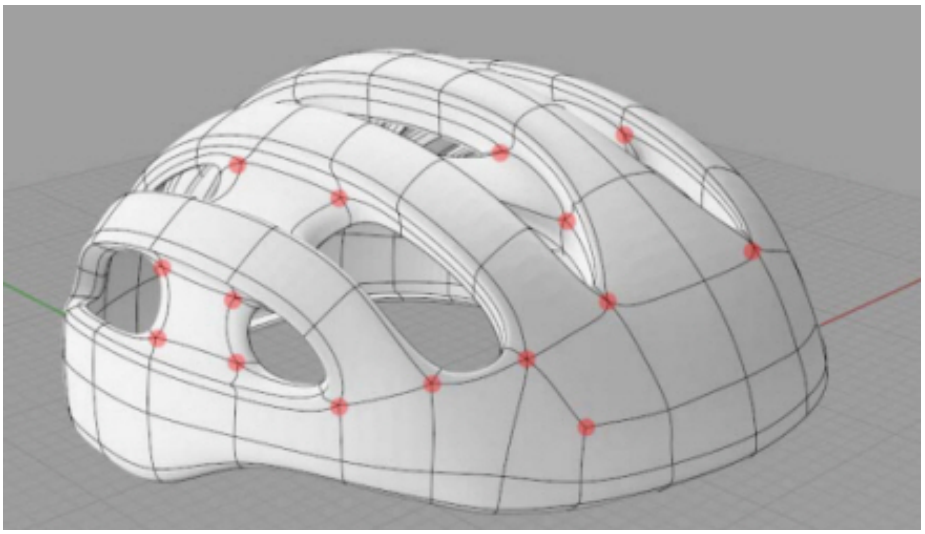

Figure 3.20: Example of a single T-spline surface designing a complex object with holes. NURBS would require multiple surfaces or polysurfaces.

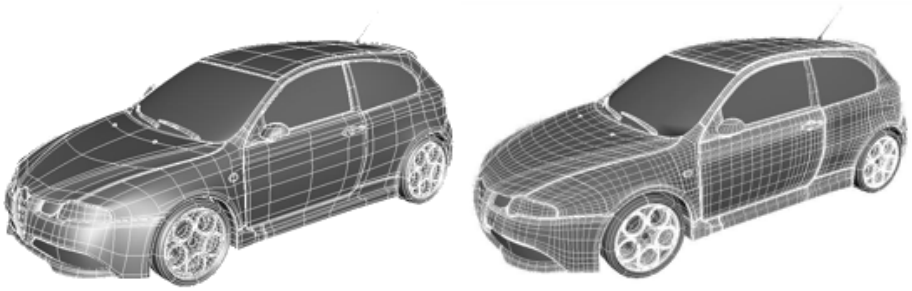

Figure 3.21: On the left car model designed using T-splines, on the right the same exact model is converted to a NURBS-based model.

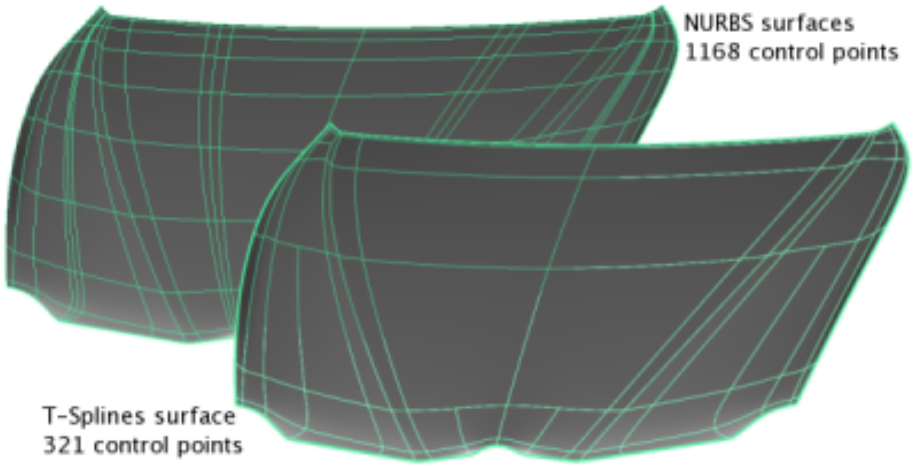

Figure 3.22: Comparison of the number of control points in a NURBS-based and in a T-spline-based model.

Chapter 4

Isogeometric Analysis

Recently, a new proposal has been developed and published in [3, 18]. The *Isogeometric Analysis* is a generalization of the Finite Element Method: while the latter employs only piecewise-polynomials for both the description of the geometry and the approximation of the solution, the former replaces them with different elements.

The starting point of the analysis of the benefits of the two is the analysis of the engineering design industry. The design description of objects and structures is embodied in Computer Aided Design systems (CAD), which has to be translated to an analysis-suitable geometry for mesh generation and use in Finite Element Analysis. This translation is, however, difficult, and turns out to need a considerable part of the overall process of analysis. After years of usage of Finite Element Analysis, it can be stated that design and analysis are not separable endeavors. However, many attempts at integrating CAD and FEM have failed.

The translation of the CAD model to a FEM suitable model is not a trivial task, and the creation of a mesh implies the introduction of a relevant approximation to the exact CAD model, which can result in analytical errors. The possibility of mesh refinement is of course available, as it can be seen from Chapter 2, but this requires a continuous and automatic communication with the CAD model, which is often not possible in the industry environment. Taking advantage of parallelism in computing the transformation of the CAD model is not simple as it is not known how to manage concurrency in mesh generation. A possible way of overcoming to these problems with Finite Element Analysis is to redesign the entire process of analysis, building one model only where it is possible to design and analyze. This is of course a huge change and requires a new analysis model based on the exact CAD representation. These are the concepts and reasons which lie behind Isogeometric Analysis, which is based on the same geometry representation employed in CAD models. The most used technology in CAD design is NURBS, but it would be possible to adapt the model to many other technologies, like T-splines.

Finite element analysis	Shared concepts	Isogeometric analysis
Nodal points		Control points
Nodal variables		Control variables
Mesh		Knots
Basis interpolates nodal points and variables		Basis does not interpolate control points and variables
Approximate geometry		Exact geometry
Polynomial basis		CAD basis
Gibbs phenomena		Variation diminishing
Subdomains		Patches
	Compact support	
	Partition of unity	
	Isoparametric concept	

Table 4.1: Comparison of the elements of FEM and Isogeometric Analysis.

4.1 FEM and Isogeometric analysis

4.1.1 General framework

Let's summarize again the model we're analyzing, already presented in Chapters 1 and 2. Consider again a second order elliptic PDE on the domain Ω with Lipschitz-continuous boundary $\Gamma = \Gamma_D \cup \Gamma_N$. The equation we would like to solve for u can be written as in Equation (1.10).

From the strong formulation, the weak formulation can be derived as in Equation (1.11).

The Galerkin projection replaces the infinite-dimensional space V by the finite-dimensional subspace V_h spanned by N_h basis functions φ_i, $i = 0, 1, \ldots, N_h - 1$. This way, the approximate solution must satisfy

$$a(v_h, \varphi_i) = l(\varphi_i), \ \forall \varphi_i \in V_h.$$

Being the space V_h linear, it is possible to write v_h as a linear combination $v_h = \sum_{i=0}^{N_h-1} \bar{v}_i \varphi_i$, so that we obtain the linear system $\boldsymbol{S}_h \cdot \bar{\boldsymbol{U}}_h = \boldsymbol{F}_h$ where $\boldsymbol{S}_h = [a(\varphi_j, \varphi_i)]_{i,j=0}^{N_h-1}$ and $\boldsymbol{F}_h = [l(\varphi_i)]_{i=0}^{N_h-1}$.

4.1.2 Isoparametric FEM

A possible choice for the basis functions, as already showed, are the piecewise-polynomials. When this is the choice, and the space of the weighting functions is equal to the set of the trial solutions, the Galerkin method is named FEM. The isoparametric concept is invoked and the boundaries of the nodal finite elements are approximated using the same basis functions used in the approximation of the solution. The domain is approximated and divided in nonoverlapping elements over which the basis functions are defined. Integrations are then performed over the reference element, where adequate integration nodes are defined.

As already illustrated in Chapter 2, possible refinements for the FEM approach are h-refinement, which refines the mesh over the domain, p-refinement,

which elevates the degree of the nodal basis functions and *hp*-refinement, which both elevates the degree and refines the mesh.

4.1.3 Isogeometric approach

Another possible choice for the shape functions are the CAD basis functions, such as B-splines, NURBS's, T-splines etc... (the important elements can be found in Chapter 3).

As already pointed, Isogeometric Analysis is an attempt at merging the engineering design and analysis: this is why the first step of Isogeometric Analysis is the description of the geometry of the problem through an "exact" description, which is the same used in the CAD model. According to the isoparametric concept, assuming B-splines are used to represent the geometry with a CAD software, the same B-splines are used in the analysis, thus allowing the use of a geometry which is exactly the same as that of the CAD model. One concept has to be remarked: it is usually not possible to use directly the CAD model in the analysis. The CAD description usually represents the boundaries of the geometry, whereas *the complete computational domain has to be included in the representation* for the analysis, as solution fields are to be computed. This means that, for two-dimensional problems, the analysis needs a surface, and not the only boundary; for a three-dimensional problem, the entire solid is needed, not only the surfaces of the solid.

The basis functions are defined over what we call the *parametric space*, which is typically the unit segment $[0,1]$ in one-dimensional problems, the unit interval $[0,1]^2$ in two-dimensional problems and the unit cube $[0,1]^3$ in three-dimensional problems. It would be very comfortable to define the shape functions used in the analysis using the same CAD basis functions, but defined on the physical space. This is, indeed, possible by using a geometrical map $\tilde{\boldsymbol{x}} : \tilde{\Omega} = [0,1]^d \to \Omega$, where d is the dimension of the CAD basis functions used in the description of the geometry. Assuming B-splines are used, for instance, $\tilde{\boldsymbol{x}}(\boldsymbol{\xi})$ is an element of the B-spline space $\mathcal{S}^{p_{\Xi_1},\dots,p_{\Xi_d}}_{\Xi_1,\dots,\Xi_d}$; so, according to the definitions of Section 3.4, in the one-dimensional case we may write the geometrical map as

$$\tilde{\boldsymbol{x}}(\xi) = \sum_{i=0}^{n} \tilde{N}_i^p(\xi)\, \boldsymbol{P}_i \in \mathcal{S}^p_{\Xi},\ \xi \in \tilde{\Omega} = [0,1] \subset \mathbb{R},$$

where the B-spline paraphernalia is defined as usual: the $\{\boldsymbol{P}_i\}_{i=0}^n$'s are the control points, the $\{w_i\}_{i=0}^n$'s are the corresponding weights and the B-spline basis functions $\tilde{N}_i^p(\xi)$'s used when expressing the $\tilde{R}_i^p(\xi)$'s are defined over the knot vector

$$\Xi = \{\xi_0 = 0, \xi_2, \dots, \xi_{n+p}, \xi_{n+p+1} = 1\}.$$

Remark 4.1. From now on, the CAD basis functions are going to be defined with a overlying tilde to attain some kind of correlation in notation with the shape functions defined on the reference domain of Chapter 2. When the CAD basis functions are written without the tilde, they denote the definition in the physical space.

According to the definitions of Section 3.4, in the two-dimensional case we

have the geometrical map

$$\tilde{\boldsymbol{x}}(\xi,\eta)=\sum_{i=0}^{n}\sum_{j=0}^{m}\tilde{N}_{i,j}^{p,q}(\xi,\eta)\,\boldsymbol{P}_{i,j}\in\mathcal{S}_{\Xi,H}^{p,q},\ \boldsymbol{\xi}=[\xi,\eta]^T\in\tilde{\Omega}=[0,1]\times[0,1]\subset\mathbb{R}^2,$$

where $n+1$ and $m+1$ are the numbers of control points in the two directions of the control net and the $N_{i,j}^{p,q}(\xi,\eta)$'s are the B-spline basis functions. The B-spline paraphernalia is defined as usual: the $\{\boldsymbol{P}_{i,j}\}_{i,j=0}^{n,m}$'s are the control points the B-spline basis functions $N_i^p(\xi)$'s and $N_j^q(\eta)$'s are defined over the knot vectors

$$\Xi=\{\xi_0=0,\xi_1,\ldots,\xi_{n+p},\xi_{n+p+1}=1\},$$
$$H=\{\eta_0=0,\eta_1,\ldots,\eta_{m+q},\eta_{m+q+1}=1\}.$$

With these tools in hand, we can define the B-spline basis functions in the physical space by using the geometrical map and their definition in the parametric space (which is simple to compute)

$$N_{i,j}^{p,q}(\boldsymbol{x})=N_{i,j}^{p,q}(x,y)=N_{i,j}^{p,q}(\tilde{\boldsymbol{x}}(\xi,\eta))=\tilde{N}_{i,j}^{p,q}(\xi,\eta)=\tilde{N}_{i,j}^{p,q}(\boldsymbol{\xi}),$$

and the solution fields can be written according to the isoparametric concept as

$$\upsilon_h(\boldsymbol{\xi})=\sum_{i=0}^{n}\sum_{j=0}^{m}\tilde{N}_i^p(\xi)\,\tilde{N}_j^q(\eta)\,\bar{\upsilon}_{i,j}.$$

The same concepts can be applied when using the NURBS basis functions to define the geometry. The map, in the two-dimensional case is, then:

$$\tilde{\boldsymbol{x}}(\xi,\eta)=\sum_{i=0}^{n}\sum_{j=0}^{m}\tilde{R}_{i,j}^{p,q}(\xi,\eta)\,\boldsymbol{P}_{i,j}\in\mathcal{N}_{\Xi,H,\boldsymbol{w}}^{p,q},\ \boldsymbol{\xi}=[\xi,\eta]^T\in\tilde{\Omega}=[0,1]\times[0,1]\subset\mathbb{R}^2.$$

Example 4.2. Suppose we want to map the parametric space $\tilde{\Omega}=[0,1]$ to the space $\Omega=[x_0=-3,x_1=5]$. Suppose moreover that the design phase has developed a model which uses the knot vector $\Xi=[0,0,1,1]$, with $p=1$. We can derive the B-spline basis functions of degree 0

$$\begin{aligned}N_0^0(\xi)&=\begin{cases}1, & 0\le\xi<0\\ 0, & \text{otherwise}\end{cases}=0,\\ N_1^0(\xi)&=\begin{cases}1, & 0\le\xi<1\\ 0, & \text{otherwise}\end{cases},\\ N_2^0(\xi)&=\begin{cases}1, & 1\le\xi<1\\ 0, & \text{otherwise}\end{cases}=0.\end{aligned}$$

and then we can build recursively the B-spline basis functions of degree 1

$$\begin{aligned}N_0^1(\xi)=\frac{\xi-0}{0-0}N_0^0(\xi)+\frac{1-\xi}{1-0}N_1^0(\xi)&=\begin{cases}1-\xi, & 0\le\xi<1\\ 0, & \text{otherwise}\end{cases},\\ N_1^1(\xi)=\frac{\xi-0}{1-0}N_1^0(\xi)+\frac{1-\xi}{1-0}N_2^0(\xi)&=\begin{cases}\xi, & 0\le\xi<1\\ 0, & \text{otherwise}\end{cases}.\end{aligned}$$

The B-spline basis functions can be used to build the NURBS basis functions (in case the initial model used NURBS's) which are the shape functions defined on the parametric space (assuming all weights equal to 1 we get the B-spline basis functions):

$$R_0^1(\xi) = \frac{N_0^1}{\sum_{\hat{i}=0}^{1} N_{\hat{i}}^1(\xi)} = \begin{cases} 1-\xi, & 0 \le \xi < 1 \\ 0, & \text{otherwise} \end{cases},$$

$$R_0^1(\xi) = \frac{N_0^1}{\sum_{\hat{i}=0}^{1} N_{\hat{i}}^1(\xi)} = \begin{cases} \xi, & 0 \le \xi < 1 \\ 0, & \text{otherwise} \end{cases}.$$

We can finally write the geometrical map $\tilde{x} : \tilde{\Omega} \to \Omega$

$$\tilde{x}(\xi) = N_0^1(\xi)\, x_0 + N_1^1(\xi)\, x_1,$$

or using the NURBS basis functions

$$\tilde{x}(\xi) = R_0^1(\xi)\, x_0 + R_1^1(\xi)\, x_1.$$

With the geometrical map it is possible to map points from the parametric space to the physical space, and it is possible to define the shape functions on the physical space by using the definitions we already gave in the parametric space. Suppose we want to find the correspondent $\hat{x}$ of the point $\hat{\xi} = 1/2$, it is sufficient to use the geometrical map this way:

$$\hat{x} = \tilde{x}\left(\hat{\xi}\right) = R_0^1\left(\hat{\xi}\right) \cdot x_0 + R_1^1\left(\hat{\xi}\right) \cdot x_1 = -3 \cdot (1 - 1/2) + 5 \cdot (1/2) = 1.$$

The definition of the shape functions in the physical space can be found by composing the $R_i^p(\xi)$'s with the inverse of the map

$$\tilde{x}(\xi) = x_0 - \xi(x_0 - x_1),$$

which is

$$\xi(\tilde{x}) = \frac{x_0 - \tilde{x}}{x_0 - x_1}.$$

The composition yields to

$$R_0^1(x) = R_0^1(\xi) \circ \xi(x) = \frac{x - x_1}{x_0 - x_1},$$

$$R_1^1(x) = R_1^1(\xi) \circ \xi(x) = \frac{x_0 - x}{x_0 - x_1}.$$

It is simple to see that these are exactly the roof functions used in linear FEM.

Isogeometric Analysis offers, just like FEM, the possibility for h-refinement, by *knot insertion* (see 4.5.1), p-refinement, by *degree elevation* (see 4.5.2), hp-refinement and ph-refinement, which is a new possibility not offered by classical FEM.

4.2 One-dimensional problems

The first step in the resolution of a one-dimensional problem defined on the domain $\Omega = (a, b)$, is the presence of the CAD model of the geometry which comprise a set of $n+1$ control points $\boldsymbol{P}_i$, $i = 0, \dots, n$, a set of $n+1$ weights w_i, $i = 0, \dots, n$, and a knot vector

$$\Xi = \left\{ \underbrace{0, \dots, 0}_{p+1}, \xi_{p+1}, \dots \xi_n, \underbrace{1, \dots, 1}_{p+1} \right\}, \ |\Xi| = n + p + 2,$$

where p indicates the degree of the NURBS curve (suppose we're working with a NURBS model, but the same considerations can be applied to B-spline models).

Starting from the weak formulation for a one-dimensional problem, which is the same written in a more general form in (1.13):

$$a(\upsilon, \varphi) = l(\varphi), \ \forall \varphi \in H^1(\Omega)$$

with

$$a(\upsilon, \varphi) = \int_\Omega (a_1 \upsilon' \cdot \varphi' + a_0 \upsilon \varphi) \, dx$$

$$l(\varphi) = \int_\Omega (f\varphi - a_1 \gamma' \cdot \varphi' - a_0 \gamma \varphi) \, dx + \left[a_1 (\upsilon + \gamma)' \varphi \right]_a^b$$

equipped with the boundary conditions

$$\upsilon = 0, \ \forall x \in \Gamma_D$$

$$(\upsilon + \gamma)' = g_N, \ \forall x \in \Gamma_N$$

where $u = \upsilon + \gamma$, a sequence of steps is needed to get an approximation.

Knot vector (mesh) In the Isogeometric Analysis, the mesh is represented by the knot vector Ξ.

Application of the Galerkin method The Galerkin method remains almost unchanged: it is possible to express the unknown function υ as a linear combination of the basis functions of the space

$$\text{span} \left\{ R_i^p(x), \ a \le x \le b, \ i = 0, \dots, n | i \in G \backslash G_D \right\},$$

where p is the degree of the NURBS basis functions R_i^p, $i = 0, \dots, n$ defined on the physical space Ω and where

$$G_D = \left\{ i | P_i = (x_i, y_i) \, | x_i \in \Gamma_D \right\},$$

$$G = \left\{ i | \exists P_i = (x_i, y_i) \right\}.$$

Remark 4.3. It is important to note that the basis functions are defined in the physical space, and not in the parametric space.

The approximate solution v can then be written as the linear combination

$$v_h(x) = \sum_{i \in G \setminus G_D} \bar{v}_i R_i^p(x).$$

A possible choice for γ is

$$\gamma(x) = \sum_{i \in G_D} g_D(P_i) R_i^p(x). \tag{4.1}$$

By substitution we get:

$$a\left(R_i^p, R_j^p\right) = \int_\Omega \left(a_1 \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_j^p}{\partial x} + a_0 R_i^p R_j^p \right) dx, \tag{4.2}$$

$$l\left(R_j^p\right) = \int_\Omega f R_j^p dx - \int_\Omega a_1 \sum_{i \in G_D} g_D(P_i) \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_j^p}{\partial x} dx +$$

$$- \int_\Omega a_0 \sum_{i \in G_D} g_D(P_i) R_i^p R_j^p dx + \left[a_1 g_N R_i^p\right]_a^b. \tag{4.3}$$

The system of algebraic equations is written with the following stiffness matrix and force vector

$$\boldsymbol{S}_h = \left[a\left(R_i^p, R_j^p\right)\right]_{i,j \in G \setminus G_D}, \tag{4.4}$$

$$\boldsymbol{F}_h = \left[l\left(R_j^p\right)\right]_{j \in G \setminus G_D}, \tag{4.5}$$

and the vector of unknowns

$$\bar{\boldsymbol{\Upsilon}}_h = [\bar{v}]_{i \in G \setminus G_D}.$$

It can be seen that Neumann and Dirichlet boundary conditions are imposed just like in FEM. Our choice of γ in Equation (4.1) is possible only for open knot vectors, for which the solution field interpolates the degrees of freedom at the extremity of the patch (3.5.4). This is possible only for boundary control points as the NURBS basis functions reach the unity only at the boundary.

Example 4.4. Consider the problem of Example 2.23: the exact solution and the weak formulation (Equation (2.14)) has already been derived. Considering the knot vector

$$\Xi = \left\{0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1\right\}$$

and basis functions of $p^{\text{th}} = 1^{\text{st}}$-degree, Isogeometric Analysis turns out to be equivalent to the Finite Element Method, as the basis functions employed are the roof functions defined in (2.10).

Example 4.5. Consider the problem of Example 2.23: the exact solution and the weak formulation of the problem have already been derived. Isogeometric Analysis can be used instead of Finite Element Method to get an approximation of the result. Using the knot vector

$$\Xi_1 = \{0, 0, 0, 1, 1, 1\}$$

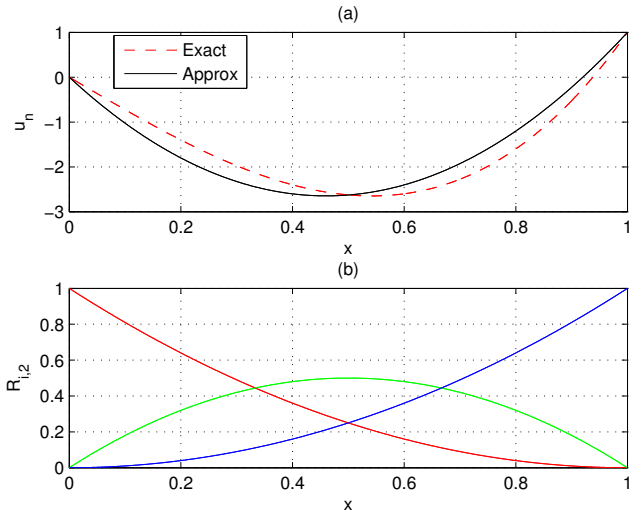

Figure 4.1: (a) Representation of an approximation (black curve) of the solution of the problem of Example 2.23 whose exact solution is plotted in red. The approximation is obtained using the knot vector $\Xi_1 = \{0, 0, 0, 1, 1, 1\}$. (b) NURBS basis functions used for the approximation.

and NURBS basis functions on Ξ of 2^{nd}-degree, the approximation of Figure 4.1 can be achieved.

It is possible to use different knot vectors in order to obtain a more accurate approximation in specific areas. In Figure 4.2 the knot vector

$$\Xi_2 = \{0, 0, 0, 0.1, 0.2, 0.3, 1, 1, 1\}$$

is used. It can be seen that the leftmost part of the approximated curve is more accurate as the space used to represent the approximation is larger. In case the same number of knots is better distributed on the domain like in

$$\Xi_3 = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\},$$

the entire curve is more accurate than the case of Figure 4.1, at the cost of a less accurate leftmost part respect to the case of Figure 4.2.

Algorithm 4.1 presents an algorithm for the computation of the DOFs in one-dimensional problems.

4.2.1 Transformation of the model to the parametric space

The case presented in Section 4.2 can be simply solved evaluating the integrals in Equations 4.4 and 4.5. It is possible a similar process to that developed in Subsection 2.3.2. The reference element in this case is the unit interval $[0, 1]$, the unit square $[0, 1]^2$ or the unit cube $[0, 1]^3$: the difference is that the entire geometry is mapped in these elements, and not only single elements of the physical space.

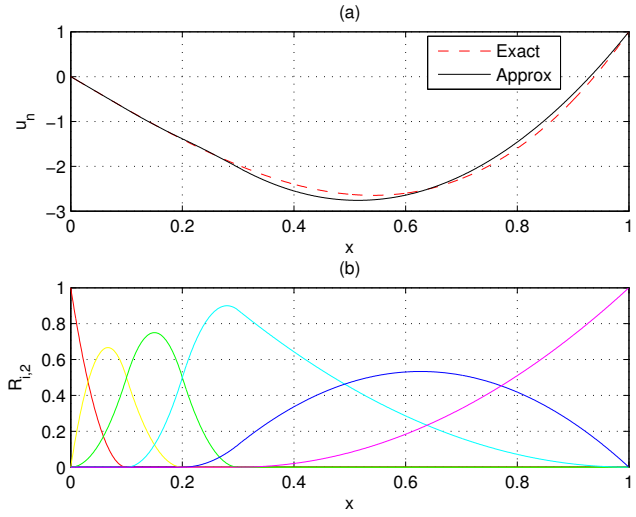

Figure 4.2: (a) Representation of an approximation (black curve) of the solution of the problem of Example 2.23 whose exact solution is plotted in red. The approximation is obtained using the knot vector $\Xi_2 = \{0, 0, 0, 0.1, 0.2, 0.3, 1, 1, 1\}$. (b) NURBS basis functions used for the approximation.

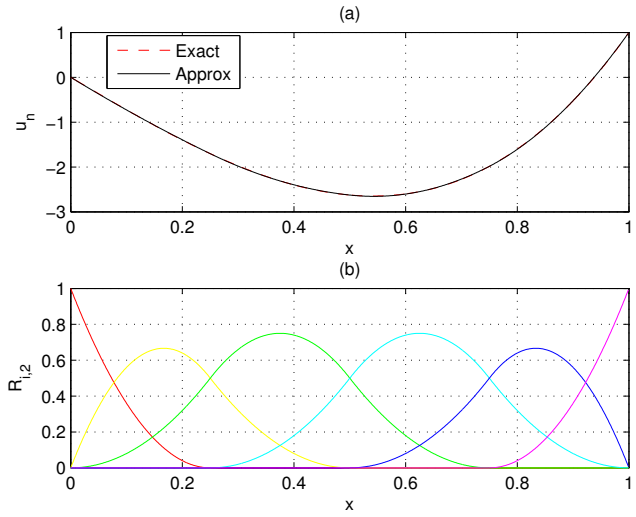

Figure 4.3: (a) Representation of an approximation (black curve) of the solution of the problem of Example 2.23 whose exact solution is plotted in red. The approximation is obtained using the knot vector $\Xi_3 = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. (b) NURBS basis functions used for the approximation.

Algorithm 4.1 Algorithm for the computation of the DOFs through IGA in a one-dimensional problem.

```
function [S, F, u] = IA1DBsplines(a_1, a_0, f, xi_a, xi_b, Xi, p)
a = Xi(1);
b = Xi(end);
m = length(Xi) - 1;
for i = 1:(m-p-2)
    for j = 1:(m-p-2)
        integrand = @(y) stiffnessIntegral(a_1, p, Xi, i, j, y);
        S(i, j) = quad(integrand, a, b);
    end
    integrand = @(y) forceIntegral(a_1, f, xi_a, xi_b, p, Xi, i, y);
    F(i, 1) = quad(integrand, a, b);
end
S
F
u = linsolve(S, F);

function s = stiffnessIntegral(a_1, p, Xi, i, j, y)
for k = 1:length(y)
    derivs1 = derivsBasisFun(p, Xi, i, y(k), 1);
    derivs2 = derivsBasisFun(p, Xi, j, y(k), 1);
    s(k) = a_1(y(k)).*derivs1(2).*derivs2(2);
end

function force = forceIntegral(a_1, f, xi_a, xi_b, p, Xi, i, y)
for k = 1:length(y)
    derivs = derivsBasisFun(p, Xi, i, y(k), 1);
    N = basisFun(p, length(Xi)-1, Xi, i, y(k));
    force(k) = f(y(k)).*N -...
        a_1(y(k)).*dgamma(xi_a, xi_b, p, Xi, y(k)).*derivs(2);
end

function dg = dgamma(xi_a, xi_b, p, Xi, y)
for k = 1:length(y)
    derivs1 = derivsBasisFun(p, Xi, 0, y(k), 1);
    derivs2 = derivsBasisFun(p, Xi, length(Xi)-p-2, y(k), 1);
    dg(k) = xi_a.*derivs1(2) + xi_b.*derivs2(2);
end
```

Transformation of functions to the parametric space Functions can be transformed to the parametric space by composing them with the geometrical map $\tilde{x}$:

$$\tilde{a}_l(\xi) = (a_l \circ \tilde{x})(\xi) = a_l(\tilde{x}(\xi)),\ l = 0, 1$$

$$\tilde{R}_l^p(\xi) = (R_l^p \circ \tilde{x})(\xi) = R_l^p(\tilde{x}(\xi)),\ l = 0, \ldots, n,$$

$$\tilde{f}(\xi) = (f \circ \tilde{x})(\xi) = f(\tilde{x}(\xi)).$$

Transformation of derivatives to the parametric space The transformation of the derivatives can be found using the chain rule

$$\frac{\partial \tilde{R}_l^p}{\partial \xi}(\xi) = \frac{\partial (R_l^p \circ \tilde{x})}{\partial \xi}(\xi) = \frac{\partial R_l^p}{\partial \xi}(\tilde{x}(\xi)) \cdot \frac{\partial \tilde{x}}{\partial \xi}(\xi),\ l = 0, \ldots, n.$$

For future use, the term $(\partial \tilde{x}/\partial \xi)(\xi)$ will be denoted with $J_{\tilde{x}}(\xi)$, which in the one-dimensional case is equal to it's determinant $|J_{\tilde{x}}(\xi)|$.

Transformation of integrals to the parametric space The last part of the process consists in the transformation of the integrals of the linear and of the bilinear forms of Equations (4.2) and (4.3)

$$a\left(R_i^p, R_j^p\right) = \int_\Omega \left(a_1 \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_j^p}{\partial x} + a_0 R_i^p R_j^p\right) dx,$$

$$l\left(R_j^p\right) = \int_\Omega f R_j^p dx - \int_\Omega a_1 \sum_{i \in G_D} g_D(P_i) \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_j^p}{\partial x} dx +$$

$$- \int_\Omega a_0 \sum_{i \in G_D} g_D(P_i) R_i^p R_j^p dx + \left[a_1 g_N R_i^p\right]_a^b.$$

to the parametric space. This step can be done by using the substitution theorem (see Section B.3), as already done in Sub subsection 2.3.2.2, on the bilinear form

$$\begin{aligned} a\left(R_i^p, R_j^p\right) &= \int_{\tilde{x}(\bar{\Omega})} \left(\left(a_1 \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_J^p}{\partial x}\right)(x) + \left(a_0 R_i^p R_j^p\right)(x)\right) dx, \\ &= \int_{\tilde{\Omega}} \left(\left(a_1 \frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_J^p}{\partial x}\right)(\tilde{x}(\xi)) + \left(a_0 R_i^p R_j^p\right)(\tilde{x}(\xi))\right) J_{\tilde{x}}(\xi)\, d\xi, \end{aligned}$$

hand on the linear form

$$l\left(R_j^p\right) = \int_{\tilde{\Omega}} \left(f R_j^p\right)(\tilde{x}(\xi))\, d\xi +$$

$$- \int_{\tilde{\Omega}} a_1(\tilde{x}(\xi)) \sum_{i \in G_D} g_D(P_i) \left(\frac{\partial R_i^p}{\partial x} \cdot \frac{\partial R_j^p}{\partial x}\right)(\tilde{x}(\xi))\, d\xi +$$

$$- \int_{\tilde{\Omega}} a_0(\tilde{x}(\xi)) \sum_{i \in G_D} g_D(P_i) \left(R_i^p R_j^p\right)(\tilde{x}(\xi))\, dx + \left[a_1(\tilde{x}(\xi)) g_N R_i^p(\tilde{x}(\xi))\right]_a^b.$$

By substituting the functions derived above, we get that the bilinear form can be evaluated in the parametric space with the equation

$$a\left(R_i^p, R_j^p\right) = \int_{\tilde{\Omega}} \left(\left(\frac{\tilde{a}_1}{J_{\bar{x}}} \frac{\partial \tilde{R}_i^p}{\partial \xi} \frac{\partial \tilde{R}_j^p}{\partial \xi} \right)(\xi) + \left(\tilde{a}_0 \tilde{R}_i^p \tilde{R}_j^p J_{\bar{x}} \right)(\xi) \right) d\xi,$$

and the linear form with the equation

$$l\left(R_j^p\right) = \int_{\tilde{\Omega}} J_{\tilde{x}} \left(\tilde{f} \tilde{R}_j^p \right)(\xi)\, d\xi - \int_{\tilde{\Omega}} \frac{\tilde{a}_1(\xi)}{J_{\tilde{x}}} \sum_{i \in G_D} g_D(P_i) \left(\frac{\partial \tilde{R}_i^p}{\partial \xi} \cdot \frac{\partial \tilde{R}_j^p}{\partial \xi} \right)(\xi)\, d\xi +$$

$$- \int_{\tilde{\Omega}} \tilde{a}_0(\xi) \sum_{i \in G_D} g_D(P_i) \left(J_{\tilde{x}} \tilde{R}_i^p \tilde{R}_j^p \right)(\xi)\, d\xi + \left[\tilde{a}_1(\xi)\, \tilde{g}_N(\xi)\, \tilde{R}_i^p(\xi) \right]_0^1.$$

4.3 Two-dimensional problems

As already pointed in Section 4.2, we assume we are given a NURBS-based model of the physical space with which we have to work. In the two-dimensional case, this means we are given a net of $(n+1)\cdot(m+1)$ control points $\boldsymbol{P}_{i,j}$, $i = 0, \ldots, n$, $j = 0, \ldots, m$, a set of $(n+1)\cdot(m+1)$ weights $w_{i,j}$, $i = 0, \ldots, n$, $j = 0, \ldots, m$, and two knot vectors

$$\Xi = \left\{ \underbrace{0, \ldots, 0}_{p+1}, \xi_{p+1}, \ldots \xi_n, \underbrace{1, \ldots, 1}_{p+1} \right\},\ |\Xi| = n + p + 2,$$

$$H = \left\{ \underbrace{0, \ldots, 0}_{q+1}, \eta_{q+1}, \ldots \eta_m, \underbrace{1, \ldots, 1}_{q+1} \right\},\ |H| = m + q + 2,$$

where p and q are respectively the degrees in the ξ and in the η direction.

Consider again the problem of solving an elliptic PDE equipped with boundary conditions on a two-dimensional domain like (1.10). Isogeometric Analysis, like the Finite Element Method, works on the weak formulation instead of working on the classical formulation. So, the same process applied in Sub-subsection 1.4.1 is necessary prior to the use of Isogeometric Analysis, and leads to the weak formulation in (1.13) or in (1.12). The key equations are reported:

$$a(v, \varphi) = l(\varphi),\ \forall \varphi \in H^1(\Omega)$$

where

$$a(v, \varphi) = \iint_{\Omega} (a_1 \nabla v \cdot \nabla \varphi + a_0 v \varphi)\, d\boldsymbol{z},\ v, \varphi \in H^1(\Omega)$$

$$l(\varphi) = \iint_{\Omega} (f\varphi - a_1 \nabla \gamma \cdot \nabla \varphi - a_0 \gamma \varphi)\, d\boldsymbol{z} + \int_{\Gamma_N} (a_1 g_N \varphi)\, d\boldsymbol{S},\ \forall \varphi \in H^1(\Omega).$$

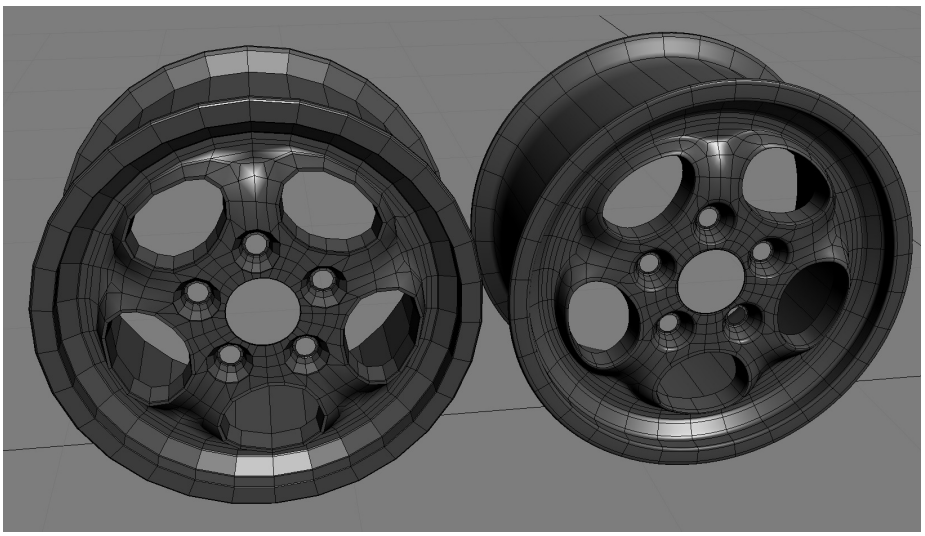

Figure 4.4: Effects of the creation of a mesh with linear elements on a real world model built using T-splines.

Approximation of the domain As explained in Section 2.4, the application of the Finite Element Method in the solution of two-dimensional problems requires some variational crimes. These variational crimes begin with the approximation of the domain Ω with another domain Ω_h, which is exact only in case the boundaries are piecewise-polynomials of the same degree of the shape functions used in the analysis. This first approximation leads to the need of approximating the boundaries and the Hilbert space. In the Isogeometric Analysis, the geometry is expressed through the use of the same CAD functions used in design, which means the exact definition is used instead of an approximated one. In Figure 4.4 it can be seen how the creation of a mesh with linear elements can affect the model which is to be analyzed.

Knot vectors (mesh) In the Isogeometric Analysis the Subdomains are called *patches* (see Table 4.1), and they are defined by the use of adequate knot vectors. It is possible to associate a mesh $\mathcal{K}$ in the parametric space to the knot vectors

$$\mathcal{K}(\Xi, H) \triangleq \{K = [\xi_i, \xi_{i+1}] \otimes [\eta_j, \eta_{j+1}] \neq \emptyset,\ i = 0, \ldots, n-1, j = 0, \ldots, m-1\}.$$

Approximation of the boundaries In the Finite Element Method the use of the approximated domain Ω_h instead of Ω can require the redefinition of the boundary conditions, as it is possible that the new boundary $\partial\Omega_h$ differs from $\partial\Omega$. In Isogeometric Analysis the representation of the geometry is exact and this means there is no more the need for an approximation of the boundary conditions as well. Here, again, no variational crime is committed.

Approximation of the Hilbert space The Hilbert space is still defined on the physical space Ω, so there is no variational crime as there was in the Finite Element Method.

Weak formulation The weak formulation remains unchanged as no variational crime has been committed.

Application of the Galerkin method The Galerkin method remains almost unchanged. As usual, it is possible to express the unknown function υ_h as the linear combination of elements of the Galerkin subspace

$$\upsilon_h(x,y) = \sum_{[i,j]\in G\backslash G_D} R_{i,j}^{p,q}(x,y)\,\bar{\upsilon}_{i,j}, \tag{4.6}$$

where the $\bar{\upsilon}_{i,j}$ are the degrees-of-freedom and G_D is now defined by the set

$$G_D = \{[i,j],\ i=0,\dots,n,\ j=0,\dots,m | \boldsymbol{P}_{i,j}\in\Gamma_D\},$$

and

$$G = \{[i,j],\ i=0,\dots,n,\ j=0,\dots,m | \exists \boldsymbol{P}_{i,j}\}.$$

Remark 4.6. It is important to note that, like in the one-dimensional case, the basis functions are defined in the physical space, and not in the parametric space. In this case, moreover, it is not simple to define these functions. This problem is addressed in 4.3.1.

The new problem is now expressed by the linear system of equations

$$\sum_{[i,j]\in G\backslash G_D}\bar{\upsilon}_{i,j}\iint_\Omega\left(a_1\nabla R_{i,j}^{p,q}\cdot\nabla R_{k,l}^{p,q}+a_0R_{i,j}^{p,q}R_{k,l}^{p,q}\right)d\boldsymbol{x}=$$

$$\iint_\Omega\left(fR_{k,l}^{p,q}-a_1\nabla\gamma\cdot\nabla R_{k,l}^{p,q}-a_0\gamma R_{k,l}^{p,q}\right)d\boldsymbol{x}+\int_{\Gamma_N}a_1g_NR_{k,l}^{p,q}d\boldsymbol{S},$$

for all $[k,l]\in G\backslash G_D$. We can again write the Dirichlet lift as

$$\gamma(\boldsymbol{x}) = \sum_{[i,j]\in G_D} R_{i,j}^{p,q}(\boldsymbol{x})\,g_D(\boldsymbol{P}_{i,j}),$$

getting the new system

$$\sum_{[i,j]\in G\backslash G_D}\bar{\upsilon}_{i,j}\iint_\Omega\left(a_1\nabla R_{i,j}^{p,q}\cdot\nabla R_{k,l}^{p,q}+a_0R_{i,j}^{p,q}R_{k,l}^{p,q}\right)d\boldsymbol{x}=$$

$$\iint_\Omega fR_{k,l}^{p,q}d\boldsymbol{x}-\iint_\Omega a_1\sum_{[i,j]\in G_D}g_D(\boldsymbol{P}_{i,j})\nabla R_{i,j}^{p,q}\cdot\nabla R_{k,l}^{p,q}d\boldsymbol{x}+$$

$$-\iint_\Omega a_0\sum_{[i,j]\in G_D}g_D(\boldsymbol{P}_{i,j})R_{i,j}^{p,q}R_{k,l}^{p,q}d\boldsymbol{x}+\int_{\Gamma_N}a_1g_NR_{k,l}^{p,q}d\boldsymbol{S},$$

for all $[k,l]\in G\backslash G_D$.

A more comfortable way of writing this, especially from the point of view of the implementation, is using one index only for each NURBS basis function, establishing a linear ordering of the basis functions. This can be done by using a map $\mathfrak{m}:\mathbb{R}\to\mathbb{R}^2$

$$\mathbf{m}(i) = [\mathfrak{m}_1(i),\mathfrak{m}_2(i)] = \left[\left\lfloor\frac{i}{m+1}\right\rfloor, i-\left\lfloor\frac{i}{m+1}\right\rfloor(m+1)\right],$$

and whose inverse is

$$\mathfrak{m}^{-1}(k,l) = k(m+1) + l.$$

With this notation, the Galerkin subspace becomes

$$\text{span}\left\{R_i^{p,q}(x,y),\ i = 0,\dots,(n+1)(m+1) \,|\, [\mathfrak{m}_1(i), \mathfrak{m}_2(i)] \in G\backslash G_D\right\},$$

and the rest of the model has to be changed accordingly:

$$\sum_{i|\mathfrak{m}(i)\in G\backslash G_D} \bar{v}_i \iint_\Omega \left(a_1 \nabla R_i^{p,q} \cdot \nabla R_j^{p,q} + a_0 R_i^{p,q} R_j^{p,q}\right) d\boldsymbol{x} =$$

$$\iint_\Omega f R_j^{p,q} d\boldsymbol{x} - \iint_\Omega a_1 \sum_{i|\mathfrak{m}(i)\in G_D} g_D\left(\boldsymbol{P}_{\mathfrak{m}(i)}\right) \nabla R_i^{p,q} \cdot \nabla R_j^{p,q} d\boldsymbol{x} +$$

$$- \iint_\Omega a_0 \sum_{i|\mathfrak{m}(i)\in G_D} g_D\left(\boldsymbol{P}_{\mathfrak{m}(i)}\right) R_i^{p,q} R_j^{p,q} d\boldsymbol{x} + \int_{\Gamma_N} a_1 g_N R_j^{p,q} d\boldsymbol{S},$$

for all $j|\mathfrak{m}(j) \in G\backslash G_D$, where it is supposed that $\boldsymbol{P}_{\mathfrak{m}(i)} = \boldsymbol{P}_{\mathfrak{m}_1(i),\mathfrak{m}_2(i)}$.

4.3.1 Transformation of the model to the parametric space

Remark 4.6 underlines the fact that the NURBS basis functions have to be defined in the physical space, and not in the parametric space (the same situation of the one-dimensional case). This makes it difficult to evaluate the integrals as a definition of such functions is not simple. This is the same difficulty encountered when trying to define the shape functions in FEM. The same concept applied in the one-dimensional case can be extended to two dimensions. We would like to use the NURBS basis functions defined in the parametric space as shape functions in the physical space.

Transformation of the functions to the parametric space Functions can be transformed to the parametric space by composing them with the NURBS geometrical map $\tilde{\boldsymbol{x}}(\boldsymbol{\xi}) = [\tilde{x}_1(\boldsymbol{\xi}), \tilde{x}_2(\boldsymbol{\xi})]^T$:

$$\tilde{a}_l(\boldsymbol{\xi}) = (a_l \circ \tilde{\boldsymbol{x}})(\boldsymbol{\xi}) = a_l(\tilde{x}_1(\boldsymbol{\xi}), \tilde{x}_2(\boldsymbol{\xi})),\ l = 0,1,$$

$$\tilde{R}_l^{p,q}(\boldsymbol{\xi}) = (R_l^{p,q} \circ \tilde{\boldsymbol{x}})(\boldsymbol{\xi}) = R_l^{p,q}(\tilde{x}_1(\boldsymbol{\xi}), \tilde{x}_2(\boldsymbol{\xi})),\ l = 0,\dots,nm,$$

$$\tilde{f}(\boldsymbol{\xi}) = (f \circ \tilde{\boldsymbol{x}})(\boldsymbol{\xi}) = f(\tilde{x}_1(\boldsymbol{\xi}), \tilde{x}_2(\boldsymbol{\xi})).$$

Transformation of derivatives to the parametric space The transformation of the derivatives can be found using the chain rule:

$$\frac{\partial \tilde{R}_l^{p,q}}{\partial \xi}(\boldsymbol{\xi}) = \left.\frac{\partial R_l^{p,q}}{\partial x}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}(\boldsymbol{\xi})} \frac{\partial \tilde{x}_1}{\partial \xi}(\boldsymbol{\xi}) + \left.\frac{\partial R_l^{p,q}}{\partial y}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}(\boldsymbol{\xi})} \frac{\partial \tilde{x}_2}{\partial \xi}(\boldsymbol{\xi}),$$

$$\frac{\partial \tilde{R}_l^{p,q}}{\partial \eta}(\boldsymbol{\xi}) = \left.\frac{\partial R_l^{p,q}}{\partial x}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}(\boldsymbol{\xi})} \frac{\partial \tilde{x}_1}{\partial \eta}(\boldsymbol{\xi}) + \left.\frac{\partial R_l^{p,q}}{\partial y}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}(\boldsymbol{\xi})} \frac{\partial \tilde{x}_2}{\partial \eta}(\boldsymbol{\xi}).$$

The result can be written in matrix form so that the Jacobian matrix can be recognized:

$$\begin{bmatrix} \dfrac{\partial \tilde{R}_l^{p,q}}{\partial \xi} \\ \dfrac{\partial \tilde{R}_l^{p,q}}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \tilde{x}_1}{\partial \xi} & \dfrac{\partial \tilde{x}_2}{\partial \xi} \\ \dfrac{\partial \tilde{x}_1}{\partial \eta} & \dfrac{\partial \tilde{x}_2}{\partial \eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial R_l^{p,q}}{\partial x} \\ \dfrac{\partial R_l^{p,q}}{\partial y} \end{bmatrix} = \left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^T \begin{bmatrix} \dfrac{\partial R_l^{p,q}}{\partial x} \\ \dfrac{\partial R_l^{p,q}}{\partial y} \end{bmatrix}.$$

$D\tilde{\boldsymbol{x}}/D\boldsymbol{\xi}$ is the Jacobian matrix of the geometric map. The searched result can be obtained by inverting the Jacobian matrix, and can be used in the computation of the other needed gradient

$$\nabla R_l^{p,q}(\boldsymbol{x}) = \left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^{-T} \nabla \tilde{R}_l^{p,q}(\boldsymbol{\xi}).$$

Transformation of the integrals to the parametric space Using the substitution theorem it is possible to transform the integrals to the parametric space. The procedure leads to the result

$$\iint_{\tilde{\boldsymbol{x}}(\tilde{\Omega})} \left(\left(a_1 \nabla R_i^{p,q} \cdot \nabla R_j^{p,q}\right)(\boldsymbol{x}) + \left(a_0 R_i^{p,q} R_j^{p,q}\right)(\boldsymbol{x})\right) d\boldsymbol{x} =$$

$$\iint_{\tilde{\Omega}} J_{\tilde{\boldsymbol{x}}} \left(\left(\tilde{a}_1 \left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^{-T} \nabla \tilde{R}_i^{p,q}\right)\left(\left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^{-T} \nabla \tilde{R}_j^{p,q}\right) + \tilde{a}_0 \tilde{R}_i^{p,q} \tilde{R}_j^{p,q}\right) d\boldsymbol{\xi},$$

and

$$\iint_{\tilde{\Omega}} J_{\tilde{\boldsymbol{x}}} \tilde{f} \tilde{R}_j^{p,q} d\boldsymbol{x} - \iint_{\tilde{\Omega}} J_{\tilde{\boldsymbol{x}}} \tilde{a}_0 \sum_{i|\mathfrak{m}(i)\in G_D} g_D\left(\boldsymbol{P}_{\mathfrak{m}(i)}\right) \tilde{R}_i^{p,q} \tilde{R}_j^{p,q} d\boldsymbol{x} +$$

$$- \iint_{\tilde{\Omega}} J_{\tilde{\boldsymbol{x}}} a_1 \sum_{i|\mathfrak{m}(i)\in G_D} g_D\left(\boldsymbol{P}_{\mathfrak{m}(i)}\right) \left(\left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^{-T} \nabla R_i^{p,q}\right) \cdot \left(\left(\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\right)^{-T} \nabla \tilde{R}_j^{p,q}\right) d\boldsymbol{x} +$$

$$+ \int_{\Gamma_N} a_1 g_N R_j^{p,q} d\boldsymbol{S}.$$

The line integral can be transformed by using another parameterize function

$$\chi(t) = (1-t)\boldsymbol{A}_1 + t\boldsymbol{A}_2.$$

The line integral can be written as

$$\int_{\Gamma_N} \left(a_1 g_N R_j^{p,q}\right)(\boldsymbol{x})\, d\boldsymbol{x} = \int_0^1 \left(\left(a_1 g_N R_j^{p,q}\right) \circ \chi(t)\right) \cdot \left\| \frac{\partial \chi(t)}{\partial t} \right\| dt,$$

but it is immediate to see we don't now the definition of $R_j^{p,q}(\boldsymbol{x})$. To determine it, it would be necessary to compute the inverse of the map $\tilde{\boldsymbol{x}}$. We would like, instead, to integrate using $\tilde{R}_j^{p,q}(\boldsymbol{\xi})$:

$$\int_{\tilde{\Gamma}_N} \phi(\boldsymbol{\xi}) \left(\tilde{a}_1 \tilde{g}_N \tilde{R}_j^{p,q}\right)(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \int_0^1 \left(\phi \tilde{a}_1 \tilde{g}_N \tilde{R}_j^{p,q}\right)(\tilde{\boldsymbol{x}}(\chi(t))) \cdot \left\| \frac{\partial \tilde{\boldsymbol{x}}(\chi(t))}{\partial t} \right\| dt,$$

where ϕ is a function which takes into account the change of geometry. By using the chain rule we derive

$$\frac{\partial \tilde{\boldsymbol{x}}\left(\chi\left(t\right)\right)}{\partial t} = \frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\left(\chi\left(t\right)\right)\frac{\partial \chi}{\partial t}.$$

We can then impose

$$\int_0^1 \left(\phi a_1 g_N R_j^{p,q}\right)\left(\tilde{\boldsymbol{x}}\left(\chi\left(t\right)\right)\right)\cdot\left\|\frac{\partial \tilde{\boldsymbol{x}}\left(\chi\left(t\right)\right)}{\partial t}\right\| dt =$$

$$\int_0^1 \left(\left(a_1 g_N R_j^{p,q}\right)\circ\chi\left(t\right)\right)\cdot\left\|\frac{\partial \chi\left(t\right)}{\partial t}\right\| dt$$

to derive that the integral can be written

$$\int_0^1 \left(\tilde{a}_1\tilde{g}_N\tilde{R}_j^{p,q}\right)\left(\chi\left(t\right)\right)\cdot\frac{\left\|\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\left(\chi\left(t\right)\right)\frac{\partial \chi}{\partial t}\right\|}{\left\|\frac{\partial \chi\left(t\right)}{\partial t}\right\|} dt =$$

$$\int_0^1 \left(\tilde{a}_1\tilde{g}_N\tilde{R}_j^{p,q}\right)\left(\chi\left(t\right)\right)\cdot\left\|\frac{D\tilde{\boldsymbol{x}}}{D\boldsymbol{\xi}}\left(\chi\left(t\right)\right)\frac{\partial\chi/\partial t}{\|\partial\chi/\partial t\|}\right\| dt.$$

This obviously assumes the Neumann boundary is transformed to a segment in the parametric space: it is possible to split this integral in more than one parameterized one-dimensional integrals when more segments are to be computed.

4.4 Approximation representing geometries and solution fields

As already pointed in 4.1.3, IGA is based on the same data produced in the design process, therefore the geometrical domain is not approximated, but it is *exact*. For instance, if degree 4 NURBS's are used in the CAD model, then the computational domain used in IGA has degree 4. In classical FEM instead, the geometrical domain is approximated using piecewise-polynomials, of degree 1 typically.

There is however a major difference in the CAD representation and the representation needed in IGA: usually, in a CAD representation, one is interested only in showing the boundaries of the elements, so that only the skin of the domain is represented. For instance, when representing a three-dimensional solid, three-dimensional surfaces may be sufficient. In IGA, instead, we need to compute solution fields, thus the whole computational domain has to be represented. Boundary surfaces are not sufficient when analyzing a three-dimensional solid. Unfortunately, this is not a trivial task: representing three-dimensional solids is far more difficult than representing only their skin (see [1]). Let's take for instance the solid of Figure 4.5: related data of Table 4.2 prove this is not a solid, but just a surface, and, from a geometrical representation point of view, there would be no benefit in modelling as a trivariate solid. For IGA instead this is not true.

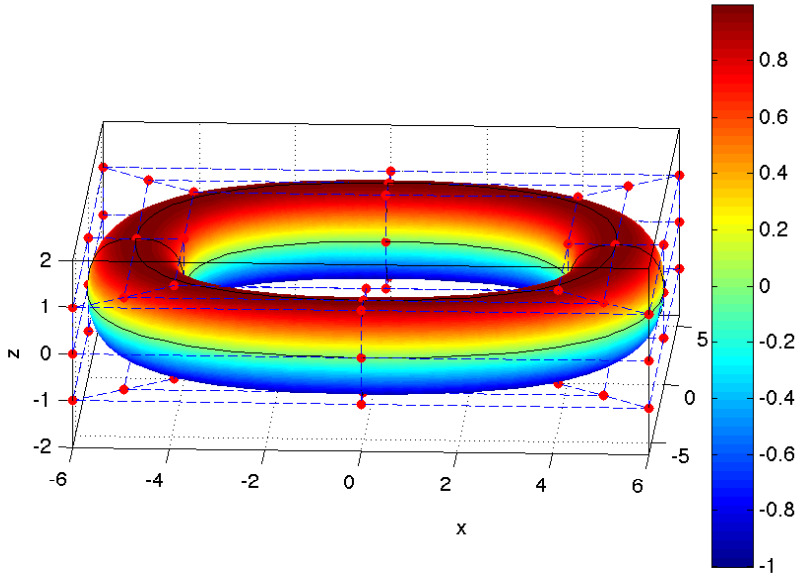

Figure 4.5: Ring plotted using Algorithm 3.9 with B-splines, using data in Table 4.2.

i	$\boldsymbol{P}_{i,1}$	$\boldsymbol{P}_{i,2}$	$\boldsymbol{P}_{i,3}$	$\boldsymbol{P}_{i,4}$	$\boldsymbol{P}_{i,5}$
1	$[5,0,-1]^T$	$[6,0,-1]^T$	$[6,0,0]^T$	$[6,0,1]^T$	$[5,0,1]^T$
2	$[5,5,-1]^T$	$[6,6,-1]^T$	$[6,6,0]^T$	$[6,6,1]^T$	$[5,5,1]^T$
3	$[0,5,-1]^T$	$[0,6,-1]^T$	$[0,6,0]^T$	$[0,6,1]^T$	$[0,5,1]^T$
4	$[-5,5,-1]^T$	$[-6,6,-1]^T$	$[-6,6,0]^T$	$[-6,6,1]^T$	$[-5,5,1]^T$
5	$[-5,0,-1]^T$	$[-6,0,-1]^T$	$[-6,0,0]^T$	$[-6,0,1]^T$	$[-5,0,1]^T$
6	$[-5,-5,-1]^T$	$[-6,-6,-1]^T$	$[-6,-6,0]^T$	$[-6,-6,1]^T$	$[-5,-5,1]^T$
7	$[0,-5,-1]^T$	$[0,-6,-1]^T$	$[0,-6,0]^T$	$[0,-6,1]^T$	$[0,-5,1]^T$
8	$[5,-5,-1]^T$	$[6,-6,-1]^T$	$[6,-6,0]^T$	$[6,-6,1]^T$	$[5,-5,1]^T$
9	$[5,0,-1]^T$	$[6,0,-1]^T$	$[6,0,0]^T$	$[6,0,1]^T$	$[5,0,1]^T$

Table 4.2: Data for ring of Figure 4.5 (continues to Table 4.3).

i	$\boldsymbol{P}_{i,6}$	$\boldsymbol{P}_{i,7}$	$\boldsymbol{P}_{i,8}$	$\boldsymbol{P}_{i,9}$
1	$[4,0,1]^T$	$[4,0,0]^T$	$[4,0,-1]^T$	$[5,0,-1]^T$
2	$[4,4,1]^T$	$[4,4,0]^T$	$[4,4,-1]^T$	$[5,5,-1]^T$
3	$[0,4,1]^T$	$[0,4,0]^T$	$[0,4,-1]^T$	$[0,5,-1]^T$
4	$[-4,4,1]^T$	$[-4,4,0]^T$	$[-4,4,-1]^T$	$[-5,5,-1]^T$
5	$[-4,0,1]^T$	$[-4,0,0]^T$	$[-4,0,-1]^T$	$[-5,0,1]^T$
6	$[-4,-4,1]^T$	$[-4,-4,0]^T$	$[-4,-4,-1]^T$	$[-5,-5,-1]^T$
7	$[0,-4,1]^T$	$[0,-4,0]^T$	$[0,-4,-1]^T$	$[0,-5,-1]^T$
8	$[4,-4,1]^T$	$[4,-4,0]^T$	$[4,-4,-1]^T$	$[5,-5,-1]^T$
9	$[4,0,1]^T$	$[4,0,0]^T$	$[4,0,-1]^T$	$[5,0,-1]^T$

Table 4.3: Data for ring of Figure 4.5 (continues from 4.2).

According to the isoparametric paradigm, solution fields are also represented using the same basis functions used in the representation of the geometry (i.e. CAD basis functions). For a two-dimensional problem we have:

$$u_h(\xi,\eta) = \sum_{i=0}^{n}\sum_{j=0}^{m} \tilde{N}_i^p(\xi)\,\tilde{N}_j^q(\eta)\,\bar{u}_{i,j}, \tag{4.7}$$

whereas for a three-dimensional problem we have

$$u_h(\xi,\eta,\zeta) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} \tilde{N}_i^p(\xi)\,\tilde{N}_j^q(\eta)\,\tilde{N}_k^r(\zeta)\,\bar{u}_{i,j,k}. \tag{4.8}$$

These ways of expressing the approximated solution are similar to that of FEM of Equation 2.7. We are using B-spline basis functions as they are more simple to manage, but NURBS and T-spline basis functions could be used as well, and they turn out to have some other interesting features. There are some differences between Lagrange interpolating polynomials and B-spline basis functions:

- B-spline basis functions are *always positive*;
- continuity along edges is not always C^0 (this will be used in 4.6);
- for seven distinct knots, eight B-spline basis functions are defined, seven Lagrange interpolating polynomials are defined over seven nodes;
- according to the properties of B-spline and NURBS basis functions, only the first and the last basis functions have unity value, whereas this is not the case for Lagrange interpolation for which the delta conditions require unity value over each node. This result in a very important concept: B-splines doesn't interpolate, in general, the control points (the DOFs).

Remark 4.7. The last concept leads to a completely different interpretation of the meaning of the DOFs: according to Theorem 2.24 the value of a DOF was the value of the exact solution at a node in FEM. In IGA this is not true anymore: it is not possible to know the solution value at a control point $\boldsymbol{P}_{i,j}$

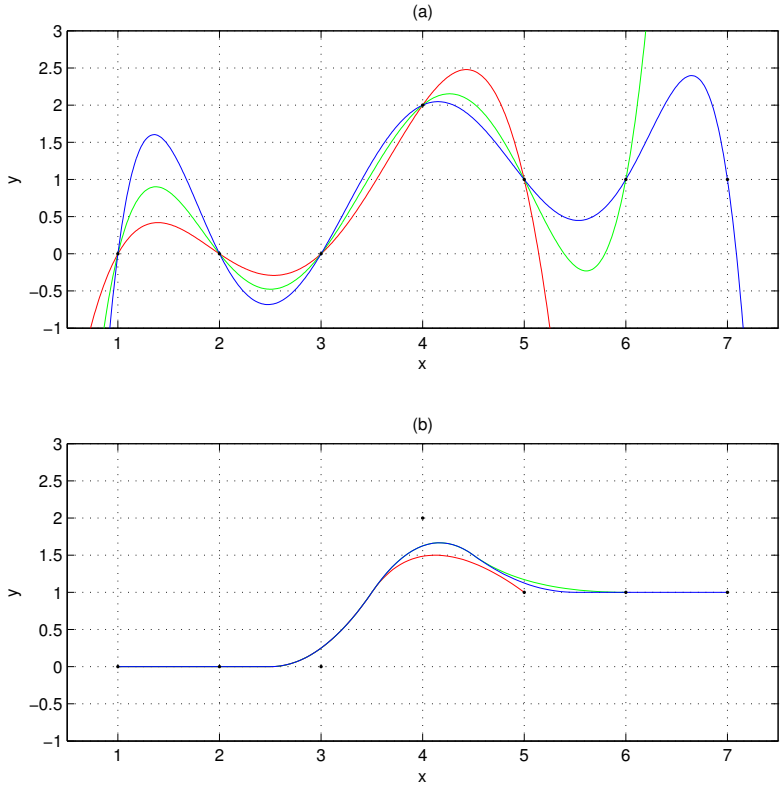

Figure 4.6: Comparison of (a) Lagrange polynomials interpolating 5, 6 and 6 points and (b) B-splines approximating the same points, taking those as control points.

or at a given knot (ξ_i, η_j) by considering only the DOF $\bar{u}_{i,j}$. The estimation of the solution value at a given point of the physical space requires Equation (4.7) or (4.8).

On the basis of Remark 4.7, a different behavior of typical finite elements functions and of CAD technologies has to be pointed out: typical finite elements functions are built to interpolate the DOFs, B-splines, NURBS's and T-splines, instead, don't interpolate DOFs in IGA. Unfortunately, it is well known that interpolating polynomials *oscillate* in attempting to fit discontinuous data and that increasing the degree of the polynomial, the amplitude of the oscillations increases as well: this is called Runge's phenomenon (see 2.3.4.2). When the points to be interpolated are used as control points instead, splines are able to approximate[1] more smoothly (see Figure 4.6). This comes from the variation diminishing property of B-splines and NURBS formulated in 3.4.4 and 3.5.4. This is very useful in case of sharp layers.

[1] Splines can both approximate and interpolate points more smoothly. In the case of IGA, interpolation is not needed as control points doesn't retain exact values like DOFs in FEM.

4.5 Refinements

As already exposed in 3.4.1 it is possible to refine the process to obtain values whose distance from the exact solution is smaller. Isogeometric Analysis differs from the Finite Element Method, as the coarsest mesh already stores all the information of the geometrical shape of the domain. Therefore, subsequent refinements don't need to communicate with the CAD data in any way. The refinement techniques described in 2.10 have corresponding refinement techniques in Isogeometric analysis, and a new alternative has been developed as well.

4.5.1 Knot insertion (h-refinement)

Considering the general form of a NURBS in (3.18) and the knot vector $\Xi = \{\xi_0, \xi_1, \dots, \xi_m\}$, knot insertion is the problem of inserting the knot $\bar{\xi} \in [\xi_k, \xi_{k+1})$ in Ξ. The new knot vector can be rewritten with the following notation:

$$\bar{\Xi} = \left\{\bar{\xi}_0 = \xi_0, \dots, \bar{\xi}_k = \xi_k, \bar{\xi}_{k+1} = \bar{\xi}, \bar{\xi}_{k+2} = \xi_{k+1}, \dots, \bar{\xi}_{m+1} = \xi_m\right\}.$$

The new NURBS is now expressed with the form

$$\bar{\boldsymbol{C}}^w(\xi) = \sum_{i=0}^{n+1} \bar{N}_i^p(\xi)\, \bar{\boldsymbol{P}}_i^{w_i}$$

where $\bar{\boldsymbol{P}}_i^{w_i}$, indicates the new i^{th} weighted control point, which needs to be determined.

Knot insertion doesn't change the curve geometrically nor parametrically (see Figure 4.7), but only adds elements to the solution space. Indeed, if $\mathcal{C}_\Xi$ is the vector space which contains all the curves representable using the knot vector Ξ and $\mathcal{C}_{\bar{\Xi}}$ is the vector space defined on $\bar{\Xi}$, then $\mathcal{C}_\Xi \subset \mathcal{C}_{\bar{\Xi}}$.

The computation of the new $\bar{\boldsymbol{P}}_i^{w_i}$'s can be computed solving the linear system of equations

$$\sum_{i=0}^{n} N_i^p(\xi)\, \boldsymbol{P}_i^{w_i} = \sum_{i=0}^{n+1} \bar{N}_i(\xi)\, \bar{\boldsymbol{P}}_i^{w_i},\ \xi = \xi_o, \dots, \xi_{n+1}.$$

However, a better way of determining the new control points is available as it can be shown that

$$\bar{\boldsymbol{P}}_i^{w_i} = \alpha_i \boldsymbol{P}_i^{w_i} + (1-\alpha_i)\, \boldsymbol{P}_{i-1}^{w_{i-1}},\ \alpha_i = \begin{cases} 1, & i \le k-p \\ \dfrac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i}, & k-p+1 \le i \le k \\ 0, & i \ge k+1 \end{cases},$$

so

$$\bar{\boldsymbol{P}}_i^{w_i} = \begin{cases} \boldsymbol{P}_i^{w_i}, & i \le k-p \\ \dfrac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} \cdot \boldsymbol{P}_i^{w_i} + \dfrac{\xi_{i+p} - \bar{\xi}}{\xi_{i+p} - \xi_i} \cdot \boldsymbol{P}_{i-1}^{w_{i-1}} & k-p+1 \le i \le k \\ \boldsymbol{P}_{i-1}^{w_{i-1}}, & i \ge k+1 \end{cases}.$$

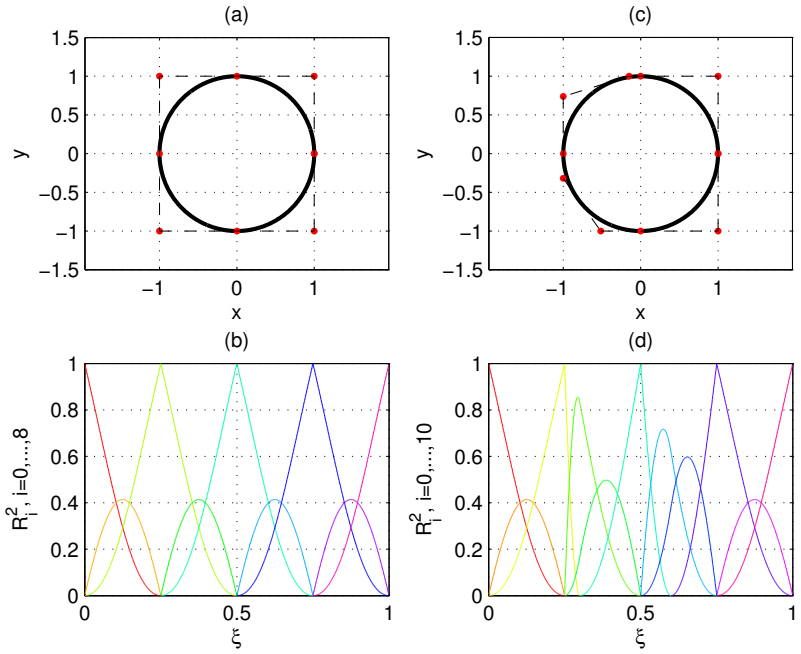

Figure 4.7: Example of knot insertion where two new knots 0.3 and 0.6 have been added to the initial knot vector.

Example 4.8. As an example, in Figure 4.7 the same circle of Example 3.16 is drawn with two new knots of values 0.6 and 0.3. As it can be seen, the resulting curve is not different from the initial, whereas both the control points and the knot vectors are different.

An efficient algorithm to perform knot insertion on a NURBS (or a B-spline) curve is reported in Algorithm 4.2.

Example 4.9. As an example, in Figure 4.8 the same surface of Example 3.17 is drawn with two new knots of values 0.7 and 0.3 in the ξ direction and three new knots of values 0.7, 0.5 and 0.3 in the η direction. As it can be seen, the resulting surface is not different from the initial, whereas both the control net and the knot vectors are different.

An efficient algorithm to perform knot insertion on a NURBS (or a B-spline) surface is reported in Algorithm 4.3.

h-refinement turns out to be particularly important when T-splines are used in Isogeometric Analysis: T-junctions allows for the insertion of control points in specific areas where the analysis needs refinement. This *local refinement* capability makes T-splines interesting both for the design and for the analysis phases.

4.5.2 Degree elevation (p-refinement)

Considering the general form of a p^{th}-degree NURBS curve $\boldsymbol{C}_p^w(\xi)$ in (3.18) defined on a knot vector Ξ, the degree elevation is the problem of computing

Algorithm 4.2 Knot insertion algorithm for curves.

```
% curveKnotIns created a new NURBS curve from an existing one in which a
% new knot barxi is inserted in [xi_k, xi_{k+1}) with multiplicity r, where
% barxi has currently multiplicity s.
% Input:
%   nP: defined accordingly to the knot vector Xi;
%   p: degree of the initial curve;
%   Xi: initial knot vector;
%   Pw: initial weighted control points;
%   barxi: value of the knot to be inserted;
%   k: index of the knot span where the knot has to be inserted;
%   s: initial multiplicity of the knot;
%   r: multiplicity of the knot in the new curve.
% Output:
%   nBarP: new value of n defined on the new knot vector;
%   barXi: new knot vector;
%   barPw: new weighted control points.
function [nBarP, barXi, barPw] = curveKnotIns(nP, p, Xi, Pw, barxi, k, s, r)
mP = nP+p+1;
nBarP = nP+r;

% Load the new knot vector.
barXi(1:k+1) = Xi(1:k+1);
barXi(k+1+1:k+r+1) = barxi;
barXi(k+1+r+1:mP+r+1) = Xi(k+1+1:mP+1);

% Save unaltered control points.
barPw(1:k-p+1, :) = Pw(1:k-p+1, :);
barPw(k-s+r+1:nP+r+1, :) = Pw(k-s+1:nP+1, :);
Rw(1:p-s+1, :) = Pw(k-p+1:k-s+1, :);

% Insert the knot r times.
for j = 1:r
    L = k-p+j;
    for i = 0:p-j-s
        alpha = (barxi-Xi(L+i+1))./(Xi(i+k+1+1)-Xi(L+i+1));
        Rw(i+1, :) = alpha.*Rw(i+1+1, :)+(1-alpha).*Rw(i+1, :);
    end
    barPw(L+1, :) = Rw(1, :);
    barPw(k+r-j-s+1, :) = Rw(p-j-s+1, :);
end

% Load remaining control points.
for i = L+1:k-s-1
    barPw(i+1, :) = Rw(i-L+1, :);
end
```

Algorithm 4.3 Knot insertion algorithm for surfaces (continues to Algorithm 4.4).

```
function [nBarP, barXi, mBarP, barEta, barPw] =...                                  1
    surfKnotIns(nP, p, Xi, mP, q, Eta, Pw, dir, knot, k, s, r)                      2
% Rearrange control points.                                                         3
Pw = permute(Pw, [1, 3, 2]);                                                        4
                                                                                    5
% Case of insertion in the xi direction.                                           6
if dir == 0                                                                         7
    % Modify the knot vectors parameters.                                           8
    nBarP = nP+r;                                                                   9
    mBarP = mP;                                                                     10
    % Load the new knot vector in xi direction.                                     11
    barXi(1:k+1) = Xi(1:k+1);                                                       12
    barXi(k+1+1:k+r+1) = knot;                                                      13
    barXi(k+1+r+1:nP+p+1+r+1) = Xi(k+1+1:nP+p+1+1);                                 14
                                                                                    15
    % Simply copy the other direction.                                            16
    barEta = Eta(1:end);                                                            17
                                                                                    18
    % Compute the alphas.                                                         19
    for j = 1:r                                                                     20
        L = k-p+j;                                                                  21
        for i = 0:p-j-s                                                             22
            alpha(i+1, j+1) = (knot-Xi(L+i+1))./(Xi(i+k+1+1)-Xi(L+i+1));            23
        end                                                                         24
    end                                                                             25
                                                                                    26
    % For each row...                                                             27
    for row = 0:mP                                                                  28
        % Save unaltered control points.                                            29
        for i = 0:k-p, barPw(i+1, :, row+1) = Pw(i+1, :, row+1); end;               30
        for i = k-s:nP, barPw(i+r+1, :, row+1) = Pw(i+1, :, row+1); end;            31
        % Load auxiliary control points.                                            32
        for i = 0:p-s, Rw(i+1, :) = Pw(k-p+i+1, :, row+1); end;                     33
        % Insert the knot r times.                                                  34
        for j = 1:r                                                                 35
            L = k-p+j;                                                              36
            for i = 0:p-j-s                                                         37
                Rw(i+1, :) = alpha(i+1, j+1).*Rw(i+1+1, :)+...                      38
                                        (1-alpha(i+1, j+1)).*Rw(i+1, :);            39
            end                                                                     40
            barPw(L+1, :, row+1) = Rw(0+1, :);                                      41
            barPw(k+r-j-s+1, :, row+1) = Rw(p-j-s+1, :);                            42
        end                                                                         43
        % Load the remaining control points.                                        44
        for i = L+1:k-s-1, barPw(i+1, :, row+1) = Rw(i-L+1, :); end;                45
    end                                                                             46
% (continues)...                                                                   47
```

Algorithm 4.4 Knot insertion algorithm for surfaces (continues from Algorithm 4.3).

```
% ...(continues)
else
    % Modify the knot vectors parameters.
    nBarP = nP;
    mBarP = mP+r;
    % Load the new knot vector in xi direction.
    barEta(1:k+1) = Eta(1:k+1);
    barEta(k+1+1:k+r+1) = knot;
    barEta(k+1+r+1:mP+q+1+r+1) = Eta(k+1+1:mP+q+1+1);

    % Simply copy the other direction.
    barXi = Xi(1:end);

    % Compute the alphas.
    for j = 1:r
        L = k-q+j;
        for i = 0:q-j-s
            alpha(i+1, j+1) = (knot-Eta(L+i+1))./(Eta(i+k+1+1)-Eta(L+i+1));
        end
    end

    % For each row...
    for row = 0:nP
        % Save unaltered control points.
        for i = 0:k-q, barPw(row+1, :, i+1) = Pw(row+1, :, i+1); end;
        for i = k-s:mP, barPw(row+1, :, i+r+1) = Pw(row+1, :, i+1); end;
        % Load auxiliary control points.
        for i = 0:q-s, Rw(i+1, :) = Pw(row+1, :, k-q+i+1); end;
        % Insert the knot r times.
        for j = 1:r
            L = k-q+j;
            for i = 0:q-j-s
                Rw(i+1, :) = alpha(i+1, j+1).*Rw(i+1+1, :)+...
                                        (1-alpha(i+1, j+1)).*Rw(i+1, :);
            end
            barPw(row+1, :, L+1) = Rw(0+1, :);
            barPw(row+1, :, k+r-j-s+1) = Rw(q-j-s+1, :);
        end
        % Load the remaining control points.
        for i = L+1:k-s-1, barPw(row+1, :, i+1) = Rw(i-L+1, :); end;
    end
end

barPw = permute(barPw, [1, 3, 2]);
```

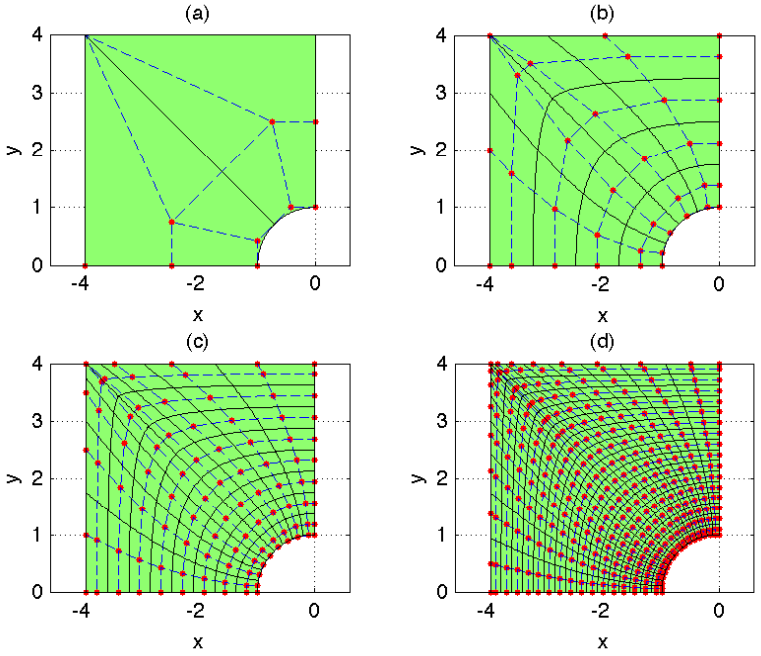

Figure 4.8: Example of knot insertion in a square plate with a hole.

the new control points $\bar{\boldsymbol{P}}_i^{w_i}$ and the new knot vector $\bar{\Xi}$ such that

$$\boldsymbol{C}_p^w(\xi) = \boldsymbol{C}_{p+1}^w(\xi) = \sum_{i=0}^{\overline{n}} N_i^{p+1}(\xi)\,\bar{\boldsymbol{P}}_i^{w_i}.$$

This way the curves continue to be the same geometrically and parametrically, but $\boldsymbol{C}_{p+1}^w$ belongs to a higher dimensional space.

Degree elevating a curve requires to compute the new knot vector and the new control points $\bar{\boldsymbol{P}}_i^{w_i}$. The continuity of the curve has to be preserved, and this is done increasing by one the multiplicity of each knot of

$$\Xi = \left\{ \underbrace{a,\ldots,a}_{p+1}, \underbrace{\xi_{p+1},\ldots,\xi_{p+1}}_{m_{p+1}}, \ldots, \underbrace{\xi_n,\ldots,\xi_n}_{m_n}, \underbrace{b,\ldots,b}_{p+1} \right\},$$

getting the vector

$$\bar{\Xi} = \left\{ \underbrace{a,\ldots,a}_{p+2}, \underbrace{\xi_{p+1},\ldots,\xi_{p+1}}_{m_{p+1}+1}, \ldots, \underbrace{\xi_n,\ldots,\xi_n}_{m_n+1}, \underbrace{b,\ldots,b}_{p+2} \right\},$$

where

$$|\Xi| = p+n+2, \quad \left|\bar{\Xi}\right| = p+n+2+(n+2).$$

The basic (and inefficient) approach to compute the $\bar{\boldsymbol{P}}_i^{w_i}$'s requires solving the linear system of equations

$$\sum_{i=0}^{\bar{n}} N_i^{p+1}(\xi)\,\bar{\boldsymbol{P}}_i = \sum_{i=0}^{n} N_i^p(\xi)\,\boldsymbol{P}_i,\ \xi = \xi_0, \ldots, \xi_{\bar{n}},$$

where the v_i's are to be chosen appropriately.

An alternative (and more efficient) algorithm is proposed in [24]: the idea is to extract Bézier curves segments from the NURBS, degree elevate them and to remove knots separating the curves to obtain the new NURBS curve.

4.5.3 $\{hp, ph\}$-refinement

As in classical FEM, it is possible to use both h-refinement and p-refinement during a single refinement process. It has been shown that this kind of refinement technique has many interesting advantages. In Isogeometric Analysis, hp-refinement can be performed as well by performing first not insertion and then degree elevating.

A new refinement method can be analyzed as knot insertion and degree elevation do not commute. This means it is possible to obtain different results by degree elevating the curve and then by inserting the new knot. ph-refinement is a different strategy of degree elevation technique which is able to achieve the same results of p-refinement, keeping the number of basis functions considerably lower.

4.6 Numerical quadrature for IGA

As already stated in Section 2.7, a relevant part of the computational time in FEM is related to the numerical integrations. In Subsections 2.4.4.2 and 2.3.2.2 we divided the integrals on the domains defined during the meshing phase of FEM. This technique was very practical in the process of assembling the matrices. A similar technique could be implemented in Isogeometric Analysis where the NURBS-based model is decomposed into a finite number of patches, and again decomposed into a grid of rectangular (in two dimensions) or parallelepipedal (in three dimensions) "elements". Again, the most obvious technique of numerical integration is the Gauss integration. The first document to analyze this problem is [19]. The following analysis will explain that this behavior could be made more efficient in Isogeometric Analysis, where the shape functions posses different continuity properties than FEM shape functions.

Both in Isogeometric Analysis and in FEM, integration has to be done over product of functions and gradients of the same functions on the parametric space and on the reference domain respectively. In Isogeometric Analysis in particular, these types of integrals are commonly computed (these reported refers to two-dimensional problems):

$$\iint_{[0,1]^2} \phi(\boldsymbol{\xi})\, R_{i,j}^{p,q}(\boldsymbol{\xi})\, R_{k,l}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{4.9}$$

$$\iint_{[0,1]^2} \phi(\boldsymbol{\xi})\, \nabla R_{i,j}^{p,q}(\boldsymbol{\xi})\, \nabla R_{k,l}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{4.10}$$

$$\iint_{[0,1]^2} \phi(\boldsymbol{\xi}) \nabla R_{i,j}^{p,q}(\boldsymbol{\xi}) R_{k,l}^{p,q}(\boldsymbol{\xi}) d\boldsymbol{\xi}, \tag{4.11}$$

$$\iint_{[0,1]^2} \phi(\boldsymbol{\xi}) \nabla R_{i,j}^{p,q}(\boldsymbol{\xi}) R_{k,l}^{p,q}(\boldsymbol{\xi}) R_{r,s}^{p,q}(\boldsymbol{\xi}) d\boldsymbol{\xi}. \tag{4.12}$$

In Equations from (4.16) to (4.19), the function $\phi(\boldsymbol{\xi})$ is a function which takes into account both the change of geometry (Substitution theorem) and the coefficients of the PDE, the $R_{i,j}$'s are the NURBS basis functions defined on the correct knot vectors and weights.

4.6.1 Numerical quadrature of C^k-continuous functions

In Chapter 2, we defined our basis with shape functions defined on the domain of the problem: these functions were piecewise-polynomials defined on the elements of the mesh. These piecewise-polynomials had continuity $C^{+\infty}$ on the elements interior, but were only C^0 on the boundaries of the elements (only continuous). Let's consider two elements, $(-1, 0)$ and $(0, 1)$, where we define a quadratic piecewise-polynomial basis with C^0 continuity on the boundaries. This basis can be analytically expressed with the six functions:

$$\begin{aligned}
\varphi_1(\xi) &= 1, \ \forall \xi \in [-1, 1], \\
\varphi_2(\xi) &= \begin{cases} -1, & \forall \xi \in [-1, 0) \\ 1, & \forall \xi \in (0, 1] \end{cases}, \\
\varphi_3(\xi) &= \xi, \ \forall \xi \in [-1, 1], \\
\varphi_4(\xi) &= \begin{cases} -\xi, & \forall \xi \in [-1, 0) \\ \xi, & \forall \xi \in (0, 1] \end{cases}, \\
\varphi_5(\xi) &= \xi^2, \ \forall \in [-1, 1], \\
\varphi_6(\xi) &= \begin{cases} -\xi^2, & \forall \xi \in [-1, 0) \\ \xi^2, & \forall \xi \in (0, 1] \end{cases}.
\end{aligned}$$

Supposing no continuity in 0, if we want to be able to integrate exactly any $\varphi \in \mathcal{S}_{2,-1} = \operatorname{span}\{\varphi_i, i = 1, \ldots, 6\}$, we need to repeat the process of Subsection 2.7.1 for the calculation of the weights and of the integration points, using the elements in $\mathcal{P}^{-1}$.

Notation 4.10. Notice that the notation of Chapter 3 of $\mathcal{S}_{\Xi}^{p}$ is replaced by $\mathcal{S}_{p,k}$ or $\mathcal{S}_{p,k}(\mathcal{M}(M))$ as, in this section it is more important to remark the continuity than the specific knot vector and the set over which the basis functions are defined.

Alternatively, we can integrate separately the two intervals using a two-point Gauss rule in each sub interval, for a total number of four integration points.

Supposing, instead, continuity C^0 in 0, we can remove $\varphi_2(\xi)$ from the basis of $\mathcal{S}_{2,-1}$ getting a basis with 5 functions. With 5 functions, only 3 integration points and weights are obtained from the system. Similarly, for C^2 continuity, $\varphi_4(\xi)$ and $\varphi_6(\xi)$ have to be removed from the basis, and only 3 functions are used to create the system, which provides 2 integration points and 2 weights only. So, the the resulting concept is that increasing the continuity reduces

k	p even	p odd
-1	$p+2$	$p+1$
even	$p+1-\frac{k}{2}$	
odd	$p+1-\frac{k+1}{2}$	

Table 4.4: Number of integration points necessary for the exact evaluation of $\int_{-1}^{1} \varphi(\xi)\, d\xi$, with $\varphi \in \mathcal{S}_{p,k}$ and $k \in ([-1, p-1] \subset \mathbb{N})$.

the number of integration points necessary to exact integrate the piecewise-polynomial. Table 4.4 summarizes these results.

All this suggests that the FEM approach to integration is not necessarily good in Isogeometric analysis as well. In this case, the integration of the spans singularly, can result in requiring more function evaluations than an integration over more spans.

Let's now consider a more general case where $\mathcal{M}_h(\mathbb{R})$ is a uniform mesh of $\mathbb{R}$ with each element $K_m \in \mathcal{M}_h(\mathbb{R})$ have length h. Functions in $\mathcal{S}_{p,-1}(\mathcal{M}_h(\mathbb{R}))$ can be evaluated with common Gauss integration on each element using n_{K_m} Gauss integration points, where $n_{K_m} = {}^{p+1}/_{2}$ when p is odd and $n_{K_m} = {}^{p+2}/_{2}$ when p is even. The exact integration can be expressed with

$$\int_{\mathbb{R}} \varphi(\xi)\, d\xi = \sum_{K_m \in \mathcal{M}_h(\mathbb{R})} \sum_{i=1}^{n_{K_m}} h w_{K_m,i} \varphi(\xi_{K_m,i}). \tag{4.13}$$

This integration is used in FEM when integrating on the different elements as functions are in $\mathcal{S}_{p,0}$, which means we need, for the case of Table 4.4, $p+1-{}^{k}/_{2} = p+1$ integration points. In Isogeometric Analysis instead, we typically have some kind of higher regularity between elements. When there is no repeated knot in the knot vector, the basis function is C^{p-1}, and so belongs to $\mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R}))$. In this particular case it can be shown that only one integration point every two elements is sufficient to get an exact integration. When p is odd the points are placed in the middle of the elements, when p is even, instead, the integration points are placed over the knots. The integration of a function $\varphi \in \mathcal{S}_{p,p-1}$ can then be written:

$$\int_{\mathbb{R}} \varphi(\xi)\, d\xi = \sum_{i \in \mathbb{Z}} h w_{\text{half-point}} \varphi(\xi_i).$$

This equation can be now used with a B-spline basis function $N_j^p(\xi) \in \mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R}))$:

$$\int_{\mathbb{R}} N_j^p(\xi)\, d\xi = \sum_{i \in \mathbb{Z}} h w_{\text{half-point}} N_j^p(\xi_i).$$

Noticing that $w_{\text{half-point}}$ is independent on p and i, it is possible to express it as

$$w_{\text{half-point}} = \frac{\int_{\mathbb{R}} N_j^p(\xi)\, d\xi}{h \sum_{i \in \mathbb{Z}} N_j^p(\xi_i)}, \tag{4.14}$$

which can be further modified considering the translation property $N_j^p(\xi) = N_0^p(\xi - jh)$ and the symmetry property $N_0^p(\xi) = N_0^p((p+1)h - \xi)$:

$$w_{\text{half-point}} = \frac{\int_{\mathbb{R}} N_0^p(\xi)\, d\xi}{h \sum_{i \in \mathbb{Z}} N_0^p(\xi_i)},$$

where $N_0^p(\xi)$ can be evaluated with the recursive definition

$$N_0^0(\xi) = \begin{cases} 1, & 0 \leq \xi \leq 1 \\ 0, & \text{otherwise} \end{cases},$$

$$N_0^p(\xi) = \frac{\xi N_0^{p-1}(\xi) + (p+1-\xi) N_0^{p-1}(\xi-1)}{p}.$$

The numerator of (4.14) can be rewritten integrating over its support $(0, (p+1)h)$:

$$\int_{\mathbb{R}} N_0^p(\xi) = \int_0^{(p+1)h} N_0^p(\xi)\, d\xi = \sum_{i=0}^{p} \int_{ih}^{(i+1)h} N_0^p(\xi)\, d\xi.$$

Again, as the support of $N_j^p(\xi)$ is $(jh, (j+p+1)h)$, the integration can be done over shifted version of the basis functions

$$\sum_{i=0}^{p} \int_{ih}^{(i+1)h} N_0^p(\xi)\, d\xi = \int_0^h \sum_{j=0}^{p} N_{-j}^p(\xi)\, d\xi$$

which, by the partition of unity property, can be simplified to

$$\int_0^h \sum_{j=0}^{p} N_{-j}^p(\xi)\, d\xi = \int_0^h 1 d\xi = h.$$

The denominator can be simplified as well: first of all the symmetry property allows to rewrite the summation

$$\begin{aligned} \sum_{i\in\mathbb{Z}} N_0^p(\xi_i) &= \frac{1}{2}\left(\sum_{i\in\mathbb{Z}} N_0^p(\xi_i) + \sum_{i\in\mathbb{Z}} N_0^p((p+1)h - \xi_i)\right), \\ &= \frac{1}{2}\left(\sum_{j\in\mathbb{Z}} N_{-j}^p(\xi_0)\right), \\ &= \frac{1}{2}. \end{aligned}$$

This means $w_{\text{half-point}} = 2$, and (4.14) becomes

$$\int_{\mathbb{R}} \varphi(\xi)\, d\xi = \sum_{i\in\mathbb{Z}} 2h\varphi(\xi_i),\ \forall \varphi(\xi) \in \mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R})). \tag{4.15}$$

(4.15) was first derived in [19] and named *half-point rule*. It has the same computational cost per degree-of-freedom as Gauss integration of Equation (4.13) for $\mathcal{S}_{p,-1}(\mathcal{M}_h(\mathbb{R}))$, which is one function evaluation every two degrees-of-freedom. The computational cost of element-wise Gauss integration would be instead much higher. Table 4.5 summarizes the results.

4.6.2 Numerical quadrature by one-dimensional integrations

We can approximate these integrals considering that the function $\phi(\boldsymbol{\xi})$ and the denominators of the NURBS basis functions change slowly, and the contributions to the integrals can often be considered constant. This result in the

Space	Gauss rule (4.13)	Half-point rule (4.15)
$\mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R}))$, p odd	$^{p+1}/_{2}$	$^{1}/_{2}$
$\mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R}))$, p even	$^{p+2}/_{2}$	$^{1}/_{2}$

Table 4.5: Number of integration points (function evaluations) for Gauss integration and half-point rule to exactly integrate element-wisely a function in $\mathcal{S}_{p,p-1}(\mathcal{M}_h(\mathbb{R}))$.

possibility of considering the integrals from (4.16) to (4.19) with the forms

$$\iint_{[0,1]^2} N_{i,j}^{p,q}(\boldsymbol{\xi})\, N_{k,l}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{4.16}$$

$$\iint_{[0,1]^2} \nabla N_{i,j}^{p,q}(\boldsymbol{\xi})\, \nabla N_{k,l}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{4.17}$$

$$\iint_{[0,1]^2} \nabla N_{i,j}^{p,q}(\boldsymbol{\xi})\, N_{k,l}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}, \tag{4.18}$$

$$\iint_{[0,1]^2} \nabla N_{i,j}^{p,q}(\boldsymbol{\xi})\, N_{k,l}^{p,q}(\boldsymbol{\xi})\, N_{r,s}^{p,q}(\boldsymbol{\xi})\, d\boldsymbol{\xi}. \tag{4.19}$$

Let's consider the case of tensor-product piecewise-quadratic C^1 basis functions: the integral of Equation (4.17), for instance, is

$$\iint_{[0,1]^2} \nabla N_{i,j}^{2,2}(\boldsymbol{\xi})\, \nabla N_{k,l}^{2,2}(\boldsymbol{\xi})\, d\boldsymbol{\xi}.$$

The two gradients can be written as

$$\nabla N_{i,j}^{2,2}(\boldsymbol{\xi}) = \left[N_j^2(\eta)\, \frac{\partial N_i^2(\xi)}{\partial \xi}, N_i^2(\xi)\, \frac{\partial N_j^2(\eta)}{\partial \eta} \right],$$

$$\nabla N_{k,l}^{2,2}(\boldsymbol{\xi}) = \left[N_l^2(\eta)\, \frac{\partial N_k^2(\xi)}{\partial \xi}, N_k^2(\xi)\, \frac{\partial N_l^2(\eta)}{\partial \eta} \right]^T,$$

and the integral, by substitution, is

$$\iint_{[0,1]^2} \left(N_j^2(\eta)\, N_l^2(\eta)\, \frac{\partial N_i^2(\xi)}{\partial \xi} \frac{\partial N_k^2(\xi)}{\partial \xi} + N_i^2(\xi)\, N_k^2(\xi)\, \frac{\partial N_j^2(\eta)}{\partial \eta} \frac{\partial N_l^2(\eta)}{\partial \eta} \right) d\eta d\xi.$$

The reduction theorem yields

$$\int_0^1 \underbrace{N_j^2(\eta)\, N_l^2(\eta)}_{\mathcal{S}_{4,1}} d\eta \int_0^1 \underbrace{\frac{\partial N_i^2(\xi)}{\partial \xi} \frac{\partial N_k^2(\xi)}{\partial \xi}}_{\mathcal{S}_{2,0}} d\xi +$$

$$+ \int_0^1 \underbrace{N_i^2(\xi)\, N_k^2(\xi)}_{\mathcal{S}_{4,1}} d\xi \int_0^1 \underbrace{\frac{\partial N_j^2(\eta)}{\partial \eta} \frac{\partial N_l^2(\eta)}{\partial \eta}}_{\mathcal{S}_{2,0}} d\eta. \tag{4.20}$$

Basis functions	Force	Stiffness	Linear adv.	Nonlinear adv.
$p=1,\ q=1,\ C^0$	$\mathcal{S}_{2,0}$	$\mathcal{S}_{0,-1},\ \mathcal{S}_{1,-1}, \mathcal{S}_{2,0}$	$\mathcal{S}_{1,-1},\ \mathcal{S}_{2,0}$	$\mathcal{S}_{2,-1},\ \mathcal{S}_{3,0}$
$p=2,\ q=2,\ C^1$	$\mathcal{S}_{4,1}$	$\mathcal{S}_{2,0},\ \mathcal{S}_{3,0}, \mathcal{S}_{4,1}$	$\mathcal{S}_{3,0},\ \mathcal{S}_{4,1}$	$\mathcal{S}_{5,0},\ \mathcal{S}_{6,1}$

Table 4.6: Sets to which the terms to be integrated belong while integrating piecewise-linear continuous basis functions and piecewise-quadratic C^1 basis functions. The force vector, stiffness matrix, linear advection and nonlinear advection terms are analyzed.

From these integrals it is clear that integrals of the form of Equation (4.17) can be evaluated exactly integrating exactly in one dimension functions in $\mathcal{S}_{4,1}$ and functions in $\mathcal{S}_{2,0}$. The same reasoning can be done for the other integrals, and the results are summarized in Table 4.6.

The considerations reported so far suggest that it would be more efficient to integrate over more than one element. In FEM, the continuity is only C^0, so no benefit would be gained integrating on more elements. Due to higher continuity, integration of B-spline basis functions is more efficient when is done over more elements. This is the reason why *macro-elements* are defined: macro-elements are made up of elements of the same size (within the same macro-element) and integrations are evaluated over them. Equation (4.15) can be used and adapted to macro-elements M_m:

$$\iint_{M_m} \varphi(\xi)\, d\xi = \sum_{i=1}^{n_{M_m}} H w_{M_m,i} \varphi(\xi_{M_m,i}),\ \forall \varphi \in \text{ a basis of } \mathcal{S}_{q,k}(\mathcal{M}_h(M_m)).$$

This equation leads again to a system of equations which has to be solved for the $2n_{M_m}$ unknowns $w_{M_m,i}$ and $\xi_{M_m,i}$. If p is the integrand function order, $r = p - k$ is the inter-element regularity and n_{el} is the number of elements in M_m then we have that (see Equation (3.4))

$$n_{dof} = (p+1) \cdot n_{el} - (p - r + 1) \cdot (n_{el} - 1)$$

is the number of degrees-of-freedom and

$$n_{M_m} = \left\lceil \frac{n_{dof}}{2} \right\rceil.$$

An algorithm is proposed in Algorithm 4.5, the algorithm for the computation of the system of nonlinear equations is reported in Algorithm 4.6. Results of these algorithms are reported in Tables (4.7)-(4.11).

4.6.3 Numerical quadrature by two-dimensional integrations

The numerical quadrature studied so far requires that the integrals can be decomposed in one-dimensional integrals using the reduction theorem. However, this is possible only when we assume the function $\phi(\boldsymbol{\xi})$ is constant. This is not frequent to happen, even when B-spline basis functions are used instead of NURBS basis functions: the determinant of the Jacobian matrix and the Jacobian matrix itself are not constant on the integration domain, and are as well not separable in (4.20).

Algorithm 4.5 Algorithm for the computation of the Gauss integration points and weights for "exactly" integrate functions in $\mathcal{S}_{p,k}([0,1])$.

```
% computeGaussPointsWeights computes the Gauss integration points and
% weigths to "exactly" integrate B-spline basis functions of degree at
% most p on a number nel of elements all of the same length where the
% continuity of the functions is C^k. The domain considere is (0,1).
% Input:
%   p: max degree of the functions to integrate "exactly";
%   nel: number of elements (elements are considered all of the same
%       length.
%   k: continuity is C^k.
% Output:
%   wxi: wxi[i], is an integration point or a weight if i is odd or even
%       respectivly.
%function wxi = computeGaussPointsWeights(p, nel, k)

% Definition of the knot vector (uniform).
Xi(1:p+1) = 0;
j = 1;
for i = p+1+1:p-k:p+1+(nel-1).*(p-k)
    Xi(i:i+p-k) = 1./(nel).*j;
    j = j+1;
end
Xi(p+1+(nel-1)*(p-k)+1:p+1+(nel-1)*(p-k)+1+p) = 1;
% Computation of the reduced continuity.
r = p-k;
% Computation of the number of degrees-of-freedom.
ndof = (p+1)*nel-(p-r+1)*(nel-1);
% computation of the number of integration points to use.
nquad = ceil(ndof/2);
% Computation of the number of knots minus one.
m = length(Xi)-1;

% Integration of the basis functions.
% Iteration on basis functions.
int(m-p) = 0;
for i = 0:m-p-1
    % Definition of the function.
    f = @(x)basisFun(p, m, Xi, i, x);
    % Evaluation of the integral.
    int(i+1) = quad(f, 0, 1, 1e-15);
end

% Definition of the nonlinear system.
F = @(x)gaussEquations(p, m, Xi, int, nquad, x);

% Setting the options for the evaluation of the nonlinear system.
options = optimset('TolX', 1e-6, 'TolFun', 1e-6,...
    'MaxFunEvals', 1e7, 'MaxIter', 1e7, 'Display', 'on');

% Solving the nonlinear system.
% ============================
% exitflag < 0 indicates the procedure has failed for some reason.
exitflag = -2;
% Initial guess of solution.
x_0(2.*nquad) = 0;
% Iterate till the system is evaluated acceptably.
while exitflag < 0
    % Define the initial guess (I want the components to be < 1 and >= 0).
    for i = 1:2.*nquad
        x_0(i) = rem(rand(1, 1), Xi(end)-Xi(1)+1) + Xi(1);
    end
    % Solution of the nonlinear system.
    [wxi, fval, exitflag, output, jacobian] = fsolve(F, x_0, options);
end
```

Algorithm 4.6 Algorithm for the computation of the value of each equation in the nonlinear system for a possible solution $\boldsymbol{x}$.

```
function [F] = gaussEquations(p, m, Xi, int, nquad, x)
% Creation of the matrix (preallocation).
F(m-p-1+1) = 0;
% Calculation of the number of the control points minus one.
n = m-p-1;

% Itarations using calculation of single basis functions.
% for j = 1:m-p-1+1
%     for k = 1:nquad
%         F(j) = F(j)+x(2.*(k-1)+1+1).*...
%             basisFun(p, m, Xi, j-1, x(2.*(k-1)+1));
%     end
% end

% Iterations using calculation of all the nonvanishing functions for each
% call.
for k = 1:nquad
    % I don't consider solutions where the integration point is outside
    % the integration domain.
    if x(2.*(k-1)+1) >= 1 ||...
            x(2.*(k-1)+1) < 0, break; end;
    % Computation of the knot span the candidate integration point
    % belongs to.
    i = findSpan(n, p, x(2.*(k-1)+1), Xi);
    % Computing all the nonvanishing B-spline basis functions in the
    % integration point.
    N = basisFuns(i, x(2.*(k-1)+1), p, Xi);
    % Evaluation of the equations (only one member).
    for j = 0:p
        F(i-j+1) = F(i-j+1)+x(2.*(k-1)+1+1).*N(p-j+1);
    end
end

% Add a symmetry condition if needed.
if m+1-p-1 < 2*nquad
    F(2*nquad) = x(ceil(nquad./2)+1) + x(nquad-ceil(nquad./2)+1+1) - 1;
end

% Subtract the "exact" integrals.
for i = 1:length(int)
    F(i) = F(i)-int(i);
end
```

$\xi_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$\xi_{[0,1),1}$	0.833333333333367	0.948645469027478	0.904711817893507	0.457777823250046
$\xi_{[0,1),2}$	0.500000000000000	0.717511861375525	0.313143439853511	0.882088086363297
$\xi_{[0,1),3}$	0.166666664870387	0.111111111111113	0.500000014840651	0.691228286634788
$\xi_{[0,1),4}$	-	0.407407407407409	0.683909215048782	0.691229590640683
$\xi_{[0,1),5}$	-	-	0.322471174843453	0.258073457384546
$\xi_{[0,1),6}$	-	-	-	0.066273731840079

$w_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$w_{[0,1),1}$	0.374999999999985	0.146083289946954	0.229417413380509	0.202575467696076
$w_{[0,1),2}$	0.250000004047672	0.282488138624475	0.496211733755255	0.270092166688278
$w_{[0,1),3}$	0.374999995952344	0.250000000000001	0.124127514678909	5131.266376438610678
$w_{[0,1),4}$	-	0.321428571428575	0.246294438348830	-5131.032497567794053
$w_{[0,1),5}$	-	-	-0.197591080045694	0.255992610234059
$w_{[0,1),6}$	-	-	-	0.170849756282850

Table 4.7: Gauss points and weights for "exact" quadrature in $[0,1)$ of functions $\mathcal{S}_{2,0}([0,1))$.

$\xi_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$\xi_{[0,1),1}$	0.166084941241376	0.923383257935759	0.320339951871245	0.402547023824487
$\xi_{[0,1),2}$	0.579519205297671	0.076616742065828	0.704204902052694	0.174032889157233
$\xi_{[0,1),3}$	0.907388611933888	0.500000000000000	0.510975818880721	0.630049724739978
$\xi_{[0,1),4}$	0.166090495942244	0.709945075524140	0.906846799509926	0.045968878372119
$\xi_{[0,1),5}$	-	0.290054924497223	0.240717961537833	0.926794576332642
$\xi_{[0,1),6}$	-	-	0.061271036140975	0.502683509307378
$\xi_{[0,1),7}$	-	-	0.273706345581009	0.300003871818280
$\xi_{[0,1),8}$	-	-	-	0.773511908662025

$w_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$w_{[0,1),1}$	- 606.535132106401875	0.181434731778163	0.439077001607745	0.071050245298913
$w_{[0,1),2}$	0.405509975080273	0.181434731855367	0.203870250601339	0.124474011028752
$w_{[0,1),3}$	0.240811371644811	0.222213207940754	0.135504983576706	0.133798040564846
$w_{[0,1),4}$	606.895868560192753	0.207458628807443	0.184264463132512	0.108857539657346
$w_{[0,1),5}$	-	0.207458628730237	0.413817312617704	0.146459106667955
$w_{[0,1),6}$	-	-	0.145238149943137	0.129856790925803
$w_{[0,1),7}$	-	-	-0.552180405364183	0.133341211220522
$w_{[0,1),8}$	-	-	-	0.141451812505181

Table 4.8: Gauss points and weights for "exact" quadrature in $[0, 1)$ of functions $\mathcal{S}_{3,0}([0, 1))$.

$\xi_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$\xi_{[0,1),1}$	0.428435167162567	0.048393309887333	0.215911159823009	0.919915886144466
$\xi_{[0,1),2}$	0.428434552289109	0.177956276631099	0.937131285756656	0.674049255828765
$\xi_{[0,1),3}$	0.121206682077936	0.500092916978055	0.215911990788857	0.080739746529892
$\xi_{[0,1),4}$	0.917131596800970	0.706031373148391	0.342198509312834	0.499834550305687
$\xi_{[0,1),5}$	0.665679754403337	0.916128622780179	0.061233362771563	0.570177698674665
$\xi_{[0,1),6}$	-	0.055906637652176	0.478517201160487	0.784021230583838
$\xi_{[0,1),7}$	-	0.317698239158576	0.625468892856977	0.368135066512957
$\xi_{[0,1),8}$	-	-	0.779573919378795	0.784164302334527
$\xi_{[0,1),9}$	-	-	0.937125207014475	0.515578174346808
$\xi_{[0,1),10}$	-	-	-	0.675049543245028
$\xi_{[0,1),11}$	-	-	-	0.226399017799747

$w_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$w_{[0,1),1}$	-0.229998305348650	0.230709827099232	-0.362634110925567	0.139577217807637
$w_{[0,1),2}$	0.500000106347717	0.159874024538251	-0.147415807665407	-0.132378555081982
$w_{[0,1),3}$	0.271526702403672	0.219895327544213	0.499999835921352	0.139885118158629
$w_{[0,1),4}$	0.199201237919740	0.194246212660173	0.137159375742357	0.402930948757977
$w_{[0,1),5}$	0.252712472333265	0.185937544149411	0.136790077641194	0.149077195992203
$w_{[0,1),6}$	-	-0.133527737279334	0.125019289354648	-0.243538802938613
$w_{[0,1),7}$	-	0.129234963447895	0.164949280922275	0.125344169690181
$w_{[0,1),8}$	-	-	0.145583932965228	0.345828566120191
$w_{[0,1),9}$	-	-	0.286837477632445	-0.352955477555863
$w_{[0,1),10}$	-	-	-	0.242691125013453
$w_{[0,1),11}$	-	-	-	0.130687525984878

Table 4.9: Gauss points and weights for "exact" quadrature in $[0, 1)$ of functions $\mathcal{S}_{4,0}([0, 1))$.

$\xi_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$\xi_{[0,1),1}$	0.042770562048058	0.536516567860769	0.625846047432515	0.458935495248663
$\xi_{[0,1),2}$	0.390281981608041	0.944080500381908	0.518075205400184	0.550323502455676
$\xi_{[0,1),3}$	0.502198842842684	0.175089395445756	0.030374840976267	0.105428992977702
$\xi_{[0,1),4}$	0.674782372288326	0.652087074128506	0.957994211814058	0.389553596573982
$\xi_{[0,1),5}$	0.194261336319405	0.408122626544791	0.233675153602255	0.186694554722707
$\xi_{[0,1),6}$	0.211213254219571	0.407358466253176	0.233669350226360	0.940298114384035
$\xi_{[0,1),7}$	0.917450573805243	0.040128526153241	0.957994100005439	0.622526382356290
$\xi_{[0,1),8}$	-	0.780866504010469	0.733780277308149	0.304845457699348
$\xi_{[0,1),9}$	-	0.175091190745680	0.132322384928325	0.024181794117765
$\xi_{[0,1),10}$	-	0.309069571522065	0.835507061935651	0.287597884998844
$\xi_{[0,1),11}$	-	-	0.430659773138450	0.727223653119565
$\xi_{[0,1),12}$	-	-	0.430821115624636	0.550284004273494
$\xi_{[0,1),13}$	-	-	0.319133784503069	0.254221673779585
$\xi_{[0,1),14}$	-	-	-	0.186742440658334
$\xi_{[0,1),15}$	-	-	-	0.822296228289488
$\xi_{[0,1),16}$	-	-	-	0.622525471004493

$w_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$w_{[0,1),1}$	0.106654746406427	0.136073617398366	0.121968600079509	0.087765889178017
$w_{[0,1),2}$	0.167378453597293	0.129036791661816	0.085178388092226	0.452614993814528
$w_{[0,1),3}$	0.087997596581449	0.347912881100316	0.074125732685454	0.092189860090221
$w_{[0,1),4}$	0.247020407398512	0.097952287825981	0.500000001781344	0.059683766666144
$w_{[0,1),5}$	0.124374267603976	-0.304512556922801	0.165203759963084	0.813297114006685
$w_{[0,1),6}$	0.068711456934242	0.417147517206252	-0.084535545806424	0.109138250470302
$w_{[0,1),7}$	0.191199446748913	0.098004632909778	-0.403111741632715	0.421519318769177
$w_{[0,1),8}$	-	0.165391841140297	0.088598218721909	0.206058167707416
$w_{[0,1),9}$	-	-0.194682693842008	0.115606170373304	0.059040102106830
$w_{[0,1),10}$	-	0.102715714062121	0.124015609557615	-0.179658803900306
$w_{[0,1),11}$	-	-	-0.228374746123211	0.106427494461614
$w_{[0,1),12}$	-	-	0.332496327089866	-0.371468049703903
$w_{[0,1),13}$	-	-	0.104324263315390	0.117476863076052
$w_{[0,1),14}$	-	-	-	-0.750539592908463
$w_{[0,1),15}$	-	-	-	0.095416934612510
$w_{[0,1),16}$	-	-	-	-0.338941011896593

Table 4.10: Gauss points and weights for "exact" quadrature in $[0, 1)$ of functions $\mathcal{S}_{6,0}([0, 1))$.

$\xi_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$\xi_{[0,1),1}$	0.646332563472504	0.785223269956529	0.072188925973708	0.142839577504758
$\xi_{[0,1),2}$	0.353667436496891	0.097993118328453	0.416461056437632	0.400172082710462
$\xi_{[0,1),3}$	0.915998404219064	0.785210003839279	0.072189051563378	0.942251167494534
$\xi_{[0,1),4}$	0.084001595757942	0.330380669998786	0.927811178643384	0.666894899108321
$\xi_{[0,1),5}$	-	0.947054076363070	0.583539908145616	0.942251232038523
$\xi_{[0,1),6}$	-	0.554407808862486	0.752693842408999	0.268055269927127
$\xi_{[0,1),7}$	-	-	0.247306319290994	0.033847598260145
$\xi_{[0,1),8}$	-	-	-	0.533396994066341
$\xi_{[0,1),9}$	-	-	-	0.802166249576074

$w_{[0,1),i}$	2 spans	3 spans	4 spans	5 spans
$w_{[0,1),1}$	0.295833814064987	- 294.320525140782877	0.409401392410386	0.120959377546950
$w_{[0,1),2}$	0.295833814389626	0.212638937321328	0.169088088044403	0.129501619152761
$w_{[0,1),3}$	0.204166185831332	294.524579768581873	-0.256628503966937	0.499999999377770
$w_{[0,1),4}$	0.204166185714050	0.224144818513231	0.152773616528585	0.135185067189173
$w_{[0,1),5}$	-	0.131264750487560	0.169088088166493	-0.377783860754304
$w_{[0,1),6}$	-	0.223357337585522	0.169660838338642	0.132287614486551
$w_{[0,1),7}$	-	-	0.169662368862370	0.082280357816187
$w_{[0,1),8}$	-	-	-	0.135083611840556
$w_{[0,1),9}$	-	-	-	0.135703312380091

Table 4.11: Gauss points and weights for "exact" quadrature in $[0, 1)$ of functions $\mathcal{S}_{4,1}([0, 1))$.

A solution is to perform the integrations on a two-dimensional space, instead of computing the product of one-dimensional integrals. This requires to create a system of equations in the form

$$\iint_{M_m} \varphi(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \sum_{i=1}^{n_{M_m}^{(\Xi)}} \sum_{j=1}^{n_{M_m}^{(H)}} H w_{M_m,i}^{(\Xi)} w_{M_m,j}^{(H)} \varphi(\xi_{M_m,i}, \eta_{M_m,j}), \tag{4.21}$$

$\forall \varphi \in$ a basis of $\mathcal{S}_{k_\Xi,k_H}^{p,q}(\mathcal{M}_h(M_m))$, where H is the length of the edge of the element. Algorithm 4.7 computes the unknowns solving the nonlinear system of Algorithm 4.9. The number of DOFs can be found in this case through Equation (3.11).

It is clear that, using the nonlinear system of equations of Equation (4.21), it is not possible to expect exact integrations for any function φ in $\mathcal{S}_{k_\Xi,k_H}^{p,q}(\mathcal{M}_h(M_m))$: this is because the points and the weights computed with Algorithms 4.7 and 4.9 didn't take into consideration the presence of a term $\phi(\boldsymbol{\xi})$. With this method, however, it no more necessary to consider $\phi(\boldsymbol{\xi})$ a constant: the stiffness term, for instance, becomes

$$\iint_{[0,1]^2} \phi(\xi,\eta) \nabla \tilde{N}_{\boldsymbol{i}}^{\boldsymbol{p}}(\xi,\eta) \nabla \tilde{N}_{\boldsymbol{j}}^{\boldsymbol{q}}(\xi,\eta)\, d\xi d\eta =$$

$$\sum_{\hat{i}=1}^{n_{M_m}^{(\Xi)}} \sum_{\hat{j}=1}^{n_{M_m}^{(H)}} w_{M_m,\hat{i}}^{(\Xi)} w_{M_m,\hat{j}}^{(H)} \left(\phi \nabla \tilde{N}_{\boldsymbol{i}}^{\boldsymbol{p}} \nabla \tilde{N}_{\boldsymbol{j}}^{\boldsymbol{q}}\right)\left(\xi_{M_m,\hat{i}}, \eta_{M_m,\hat{j}}\right).$$

The same exact concepts here reported can be applied to NURBS's.

4.7 Implementation of IGA

Possible algorithms for the implementation are reported and described here.

4.7.1 Providing design model

As usual, IGA works on a model which comes directly from the design phase. The description of the model can employ many types of CAD structures, like B-splines, NURBS's or even T-splines. In this phase, it is important to remember what has been pointed out in 4.4. In our model, we can completely describe a B-spline surface by defining:

- the degree of the B-spline basis functions (in both directions) `p` and `q`;
- the number of control points in each direction $\mathtt{n}+1$ and $\mathtt{m}+1$;
- the control points $\boldsymbol{P}_{i,j}$ which are described in the implementation as a three-dimensional matrix `P[i, j, d]` where $1 \leq \mathtt{i} \leq n+1$ and $1 \leq \mathtt{j} \leq m+1$ indicates the indices of the point and `d` is the axis to which the coordinate refers.

Once these information are defined, some functions are available to plot the physical space. The algorithms behind these functions are reported and explained in Chapter 3.

Algorithm 4.7 Algorithm for the computation of the weights and of the quadrature points for the integration of a two-dimensional function $\varphi \in \mathcal{S}^{p,q}_{k_\Xi,k_H}(\mathcal{M}_h(M_m))$ (continues to Algorithm 4.8).

```
% computeGaussPointsWeights2D computes the Gauss integration points and
% weigths to "exactly" integrate bivarite B-spline basis functions of
% degree p in direction xi and q in direction eta at on a number nel of
% uniform elements where the continuity of the functions is C^rXi in
% direction xi and C^rEta in direction eta. The domain considered is
% (0,1).
% Input:
%   p: max degree of the functions to integrate "exactly" in
%       direction xi;
%   q: max degree of the functions to integrate "exactly" in
%       direction eta;
%   k: number of elements in the xi direction;
%   l: number of elements in the eta direction;
%   nel: number of elements (elements are considered all of the same
%       length.
%   rXi: continuity is C^rXi in direction xi;
%   rEta: continuity is C^rEta in direction eta.
% Output:
%   pw: pw[i:i+4] contains, in this order, int. points on xi and eta and
%       weights on xi and on eta.
function pw = computeGaussPointsWeights2D(p, q, k, l, rXi, rEta)
% Definition of the knot vectors (uniform).
Xi(1:p+1) = 0;
j = 1;
for i = p+1+1:p-rXi:p+1+(k-1)*(p-rXi)
    Xi(i:i+p-rXi) = 1./(k).*j;
    j = j+1;
end
Xi(p+1+(k-1)*(p-rXi)+1:p+1+(k-1)*(p-rXi)+1+p) = 1;
Eta(1:q+1) = 0;
j = 1;
for i = q+1+1:q-rEta:q+1+(l-1).*(q-rEta)
    Eta(i:i+q-rEta) = 1./(l).*j;
    j = j+1;
end
Eta(q+1+(l-1)*(p-rEta)+1:q+1+(l-1)*(q-rEta)+1+q) = 1;
% Computation of the number of repetition of a knot.
sXi = p-rXi;
sEta = q-rEta;
% Computation of the number of degrees-of-freedom.
ndofXi = (p+1)*k-(p-sXi+1)*(k-1);
ndofEta = (q+1)*l-(q-sEta+1)*(l-1);
ndof = ndofXi*ndofEta;
% computation of the number of integration points to use.
nquad = ceil(ndof/2);
% Evaluation of the number of basis functions.
n = length(Xi)-p-2;
m = length(Eta)-q-2;
% (continues...)
```

Algorithm 4.8 Algorithm for the computation of the weights and of the quadrature points for the integration of a two-dimensional function $\varphi \in \mathcal{S}^{p,q}_{k_\Xi,k_H}(\mathcal{M}_h(M_m))$ (continues from Algorithm 4.7).

```
% (... continues)
% Exact integration of the basis functions.
% ========================================
int((m+1)*(n+1)) = 0;
intIndex = 1;
for i = 0:n
    for j = 0:m
        % Definition of the function.
        f = @(x, y)basisFun(p, n+p+1, Xi, i, x).*...
        basisFun(q, m+q+1, Eta, j, y);
        % Evaluation of the "exact" integral.
        for xi_i = 1:length(Xi)-1
            for eta_i = 1:length(Eta)-1
                int(intIndex) = int(intIndex)+...
                    dblquad(f, Xi(xi_i), Xi(xi_i+1),...
                    Eta(eta_i), Eta(eta_i+1));
            end
        end
        intIndex = intIndex+1;
    end
end

% Definition of the nonlinear system.
F = @(x)gaussEquations2D(p, n, Xi, q, m, Eta, int, nquad, x, k, 1/k);

% Setting the options for the evaluation of the nonlinear system.
options = optimset('TolX', 1e-3, 'TolFun', 1e-3,...
    'MaxFunEvals', 1e8, 'MaxIter', 1e8, 'Display', 'on');

% Solving the nonlinear system.
% ============================
% exitflag < 0 indicates the procedure has failed for some reason.
exitflag = -2;
% Initial guess of solution.
x_0(2.*nquad) = 0;
% Iterate till the system is evaluated acceptably.
while exitflag < 0
    % Define the initial guess (I want the components to be < 1 and >= 0).
    for i = 1:2*nquad
        x_0(i) = rem(rand(1, 1), Xi(end)-Xi(1)+1) + Xi(1);
    end
    % Solution of the nonlinear system.
    [pw, fval, exitflag, output] = fsolve(F, x_0, options)
end
```

Algorithm 4.9 Algorithm for the computation of the value of each equation in the nonlinear system for a possible solution $\boldsymbol{x}$.

```
function F = gaussEquations2D(p, n, Xi, q, m, Eta, int, nquad, x)
% Creation of the matrix (preallocation).
F((m+1)*(n+1)) = 0;

% Iterations using calculation of all the nonvanishing functions for each
% call.
for k = 1:nquad
    if x(4.*(k-1)+1) < 0 || x(4.*(k-1)+1) > 1 ||...
            x(4.*(k-1)+1+1) < 0 || x(4.*(k-1)+1+1) > 1
        F = ones(1, (m+1)*(n+1));
        return;
    end

    % Computation of the knot span the candidate integration point
    % belongs to.
    xi_i = findSpan(n, p, x(4.*(k-1)+1), Xi);
    eta_j = findSpan(m, q, x(4.*(k-1)+1+1), Eta);
    % Computing all the nonvanishing B-spline basis functions in the
    % integration point.
    Ni = basisFuns(xi_i, x(4.*(k-1)+1), p, Xi);
    Nj = basisFuns(eta_j, x(4.*(k-1)+1+1), q, Eta);
    % Evaluation of the equations (only one member).
    for a = 0:p
        for b = 0:q
            % Computation of the correct index of the current
            % nonvanishing basis functions.
            i = xi_i-a;
            j = eta_j-b;
            % Computation of the current linear indices in the F matrix.
            FIndex = i*(m+1)+j;
            F(FIndex+1) = F(FIndex+1)+...
                x((k-1)*4+2+1)*x((k-1)*4+3+1)*Ni(p-a+1)*Nj(q-b+1);
        end
    end
end

% Add a symmetry condition if needed.
for i = (n+1)*(m+1)+1:2*nquad
    if mod(i, 2) == 0
        F(i) = x(end-ceil(i/2)*3) + x(ceil(i/2)) - 1;
    else
        F(i) = x(end-ceil(i/2)*2) + x(ceil(i/2)+1) - 1;
    end
end

% Subtract the "exact" integrals.
for i = 1:length(int)
    F(i) = F(i)-int(i);
end
```

- A B-spline surface defined over adequate knot vectors `Xi` and `Eta` with `n`+1 and `m`+1 control points and degrees `p` and `q` can be evaluated in $[\texttt{xi}, \texttt{eta}]^T$ using the function `bsplineSurfPoint(n,p,Xi,m,q,Eta,P,xi,eta)`.

- A NURBS surface defined over adequate knot vectors `Xi` and `Eta` with `n` + 1 and `m` + 1 control points, degrees `p` and `q` and weights defined by a two-dimensional matrix `w` can be evaluated in $[\texttt{xi}, \texttt{eta}]^T$ using the function `NURBSSurfPoint(n,p,Xi,m,q,Eta,Pw,xi,eta)` where `Pw` are the weighted control points.

4.7.2 Providing PDE specification and boundary conditions

The problem has to be defined, comprising both the coefficients of the PDE and the boundary conditions. This can be done by using two matrices, namely **C** and **D**, defined as

$$\texttt{C}[\texttt{i},\texttt{j}] = \begin{cases} 1, & \text{if } \boldsymbol{P}_{i,j} \in \Gamma_D \\ 2, & \text{if } \boldsymbol{P}_{i,j} \in \Gamma_N \\ 0, & \text{otherwise} \end{cases}, \; \texttt{D}[\texttt{i},\texttt{j}] = \begin{cases} 0, & \text{if } \boldsymbol{P}_{i,j} \notin \Gamma_D \\ g_D\left(\boldsymbol{P}_{i,j}\right), & \text{if } \boldsymbol{P}_{i,j} \in \Gamma_D \end{cases}.$$

Neumann boundary conditions have to be applied in a weak sense. A function $g_N(\boldsymbol{x})$ has to be defined and supplied to the algorithm and the control points where a Neumann condition is imposed have to be selected in the array `C`. Moreover, for simplicity, a vector `N` is used to identify the edges on the parametric space which have Neumann conditions applied. `N` is 1 when it identifies a Neumann edge, where the first component relates to the edge with $\eta = 1$, the second to the edge with $\xi = 1$, the third to the edge with $\eta = 1$ and the fourth to the edge with $\xi = 0$. A more advanced structure for the definition of the boundary conditions may be explored however.

4.7.3 Computation of the linear system

Homogeneous and nonhomogeneous boundary conditions can be imposed directly to the control points, removing the respective equations from the system. This is done in Algorithm 4.10, where we take advantage of the symmetry of the stiffness matrix. The algorithms for the computation of the integrals will be discussed in 4.7.5. Nonhomogeneous boundary conditions can be implemented just by subtracting a term from each element of the force vector.

4.7.4 Mesh refinement

The coarsest mesh in IGA stores all the needed information, but the mesh (formed by the knot vectors) is likely to be not sufficient. Starting from the coarsest mesh, it is possible to h-refine the mesh by knot insertion. Assuming for simplicity a uniform mesh is needed, Algorithm 4.11 produces the needed result.

Algorithm 4.10 Computation of the force vector and of the stiffness matrix for nonhomogeneous Dirichlet conditions.

```
% Preallocation of the stiffness matrix and of the force vector.
S(sum(sum(C==0)), sum(sum(C==0))) = 0;
F(sum(sum(C==0))) = 0;
% Indices of the matrices.
a = 0;
b = 0;
for ki = 0:n
    for li = 0:m
        % ki and li are the indices of the first B-spline function.
        % If the node is a Dirichlet node, then skip to the next
        % iteration.
        if C(ki+1, li+1) == 1, continue; end;
        % Integration of the homogeneous part of the force vector.
        F(a+1) = forceIntegralBsplines...
            (n, p, Xi, m, q, Eta, P, ki, li, f, wp_force);
        % Iteration on the second B-spline basis function.
        for kj = 0:n
            for lj = 0:m
                % If the node belongs to the set of Dirichlet nodes then
                % it is taken into account only for the force vector.
                if C(kj+1, lj+1) == 1
                    % If the Dirichlet condition is homogeneous then
                    % it is convenient to short-circuited so that
                    % the integral is not computed.
                    if D(kj+1, lj+1) ~= 0
                        % The term is subtracted from the force vector
                        % term.
                        F(a+1) = F(a+1)-D(kj+1, lj+1).*...
                            stiffnessIntegralBsplines...
                            (n, p, Xi, m, q, Eta, P, ki, li, kj, lj,...
                            wp_stiffness, a_1);
                    end
                    continue;
                end
                % Skip the terms which can be computed by exploiting
                % symmetry.
                if b < a, b = b+1; continue; end
                ki, kj, li, lj
                % Stiffness term.
                S(a+1, b+1) = stiffnessIntegralBsplines...
                    (n, p, Xi, m, q, Eta, P, ki, li, kj, lj,...
                    wp_stiffness, a_1);
                b = b+1;
            end
        end
        a = a+1;
        b = 0;
    end
end

% Complete the lower triangle (symmetry).
for i = 1:length(S(1, :))
    for j = 1:i
        S(i, j) = S(j, i);
    end
end
```

Algorithm 4.11 Process of refinement which produces a uniform mesh with $(c-a)/b$ elements.

```
for i = a:b:c
    k = findSpan(n, p, i, Xi);
    if Xi(k+1) ~= i
        [n, Xi, m, Eta, P] =...
            surfKnotIns(n, p, Xi, m, q, Eta,...
            P, 0, i, k, 0, 1);
    end

    k = findSpan(m, q, i, Eta);
    if Eta(k+1) ~= i
        [n, Xi, m, Eta, P] =...
            surfKnotIns(n, p, Xi, m, q, Eta,...
            P, 1, i, k, 0, 1);
    end
end
```

Algorithm 4.12 Algorithm for the computation of the integral necessary to build the stiffness matrix using the concepts reported in [37] (adaptive recursive Simpson's rule).

```
function s = stiffnessIntegralBsplines(n, p, Xi, m, q,...
    Eta, P, ki, li, kj, lj, pw, a_1)
s = 0;
% According to the properties of the B-spline basis functions, each
% function is nonzero only in a specified interval. This property is useful
% in order to reduce the domain of computation of the integral.
a = max([(ki+1), (kj+1)]);
b = min([(ki+p+1+1), (kj+p+1+1)])-1;
c = max([(li+1), (lj+1)]);
d = min([(li+q+1+1), (lj+q+1+1)])-1;
% The integral is divided over the single elements as along the edges the
% degree of continuity is not known: it could lead to a wrong computation
% of the integral.
for xi_i = a:b
    for eta_i = c:d
        s = s+dblquad(@(x, y)stiffnessIntegrandBsplines...
            (n, p, Xi, m, q, Eta, P, x, y,...
            ki, li, kj, lj, a_1), Xi(xi_i), Xi(xi_i+1),...
            Eta(eta_i), Eta(eta_i+1));
    end
end
```

4.7.5 Implementations of integration in IGA

According to what has been said, two algorithms for the integration for both the stiffness matrix and the force vector are reported: Algorithms 4.12 and 4.14 integrate using adaptive recursive Simpson's rule (see [37]) and Algorithms 4.13 and 4.15 integrates using what has been proposed in 4.6. The integrands are reported in Algorithms 4.16 and 4.18.

A possible performance boost could be achieved pre-computing all the evaluations of the B-spline basis functions and of the derivatives on all the Gauss nodes.

The Neumann term has to be computed in forceIntegralBspline(...) method: the algorithms 4.19 and 4.20 shows a possible way of doing this.

Algorithm 4.13 Algorithm for the computation of the integral necessary to build the stiffness matrix using the concepts of 4.6.

```
s = 0;                                                                  1
% Iteration over all the couples of integration nodes.                 2
for i = 1:4:length(pw)-3                                                3
    s = s+pw(i+2)*pw(i+3)*stiffnessIntegrandBsplines...                 4
            (n, p, Xi, m, q, Eta, P, pw(i), pw(i+1),...                 5
            ki, li, kj, lj, a_1);                                       6
end                                                                     7
```

Algorithm 4.14 Algorithm for the computation of the integral necessary to build the force vector using the concepts reported in [37] (adaptive recursive Simpson's rule).

```
function F = forceIntegralBsplines...                                   1
    (n, p, Xi, m, q, Eta, P, ki, li, f, g_N, a_1, pw, N)                2
F = 0;                                                                  3
for xi_i = ki+1:ki+p+1                                                  4
    for eta_i = li+1:li+q+1                                             5
        F = F+...                                                       6
            dblquad(@(x, y)(forceIntegrandBsplines...                   7
            (n, p, Xi, m, q, Eta, P, x, y, ki, li, f)),...              8
            Xi(xi_i), Xi(xi_i+1),...                                    9
            Eta(eta_i), Eta(eta_i+1), 1e-5);                            10
    end                                                                 11
end                                                                     12
```

Algorithm 4.15 Algorithm for the computation of the integral necessary to build the force vector using the concepts of 4.6.

```
F = 0;                                                                  1
for i = 1:4:length(pw)-3                                                2
    F = F+pw(i+2)*pw(i+3)*forceIntegrandBsplines...                     3
        (n, p, Xi, m, q, Eta, P, pw(i), pw(i+1), ki, li, f);            4
end                                                                     5
```

Algorithm 4.16 Algorithm for the computation of the integrand for the stiffness term (continues to Algorithm 4.17).

```
function s = stiffnessIntegrandBsplines(n, p, Xi, m, q,...
    Eta, P, xi, eta, ki, li, kj, lj, a_1)
% Preallocation.
s(length(xi)) = 0;
oldeta = eta;
% Iteration on all the requested values on which to eval the integrand.
for l = 1:length(xi)
    moveForward = false;
    moveXi = false;
    moveMax = 100;
    moveCurrent = 0;
    eta = oldeta;
    Jx = 0;
    rcond = 1e-5;
    % I want to avoid the determinant of the Jacobian matrix to vanish.
    % This happens when a singularity is found and it is not possible
    % to translate the function to the parametric domain. In this case
    % I simply move by a small percentage the point in which the
    % integrand is evaluated.
    while Jx == 0
        % Determination of the derivatives. This calculation involves
        % the calculation of the value of the bsis functions in the
        % points and of the knot spans in which the points are located.
        % I can re-use these values later on, avoiding new evauations.
        [SKL, eXi, eEta, spanxi, spaneta] =...
            bsplineSurfDerivs(n, p, Xi, m, q, Eta, P, xi(l), eta, 1);
        % Definition of the jacobian matrix.
        DxDxi = [SKL(2, 1, 1), SKL(1, 2, 1); SKL(2, 1, 2), SKL(1, 2, 2)];
        % Evaluate the Jacobian.
        Jx = det(DxDxi);
        if Jx == 0, Jx, end;
        % If the Jacobian is 0, the matrix is singular and therefore not
        % invertible. This is not acceptable as I need it to be
        % invertible.
        if Jx ~= 0, break; end;
            % (continues...)
```

Algorithm 4.17 Algorithm for the computation of the integrand for the stiffness term (continues from Algorithm 4.16).

```
            % (continues...)
        % Check to see if we still have to move in the same direction.
        if moveXi == true
            % If possible move back on xi in the parametric space.
            if moveForward == true && xi(l) >= Xi(end)-rcond
                moveForward = false;
            end
            if moveForward == false && xi(l) <= Xi(1)+rcond
                moveForward = true;
            end
        else
            % If possible move back on eta in the parametric space.
            if moveForward == true && eta >= Eta(end)-rcond
                moveForward = false;
            end
            if moveForward == false && eta <= Eta(1)+rcond
                moveForward = true;
            end
        end
        moveCurrent = moveCurrent+1;
        if moveCurrent >= moveMax, moveXi = ~moveXi; end;
        if moveXi == true
            if moveForward == false, xi(l) = xi(l)-rcond;
            else xi(l) = xi(l)+rcond; end;
        else
            if moveForward == false, eta = eta-rcond;
            else eta = eta+rcond; end;
        end
    end
    % Compute the inverse of the jacobian matrix.
    DxiDx = inv(DxDxi);
    dki = ki-spanxi+p+1;
    dli = li-spaneta+q+1;
    dkj = kj-spanxi+p+1;
    dlj = lj-spaneta+q+1;
    % According to the local support property of the B-spline functions
    % they are zero outside their local support.
    if dki>0 && dki<=p+1 && dli>0 && dli<=q+1
        dNidxi = eXi(2, dki).*eEta(1, dli);
    else dNidxi = 0; end;
    if dli>0 && dli<=q+1 && dki>0 && dki<=p+1
        dNideta = eEta(2, dli).*eXi(1, dki);
    else dNideta = 0; end;
    if dkj>0 && dkj<=p+1 && dlj>0 && dlj<=q+1
        dNjdxi = eXi(2, dkj).*eEta(1, dlj);
    else dNjdxi = 0; end;
    if dlj>0 && dlj<=q+1 && dkj>0 && dkj<=p+1
        dNjdeta = eEta(2, dlj).*eXi(1, dkj);
    else dNjdeta = 0; end;
    % Definition of the gradients.
    gradNi = [dNidxi; dNideta];
    gradNj = [dNjdxi; dNjdeta];
    % Evaluation of the integrand.
    s(l) = Jx.*1.*(DxiDx'*gradNi)'*(DxiDx'*gradNj).*...
        a_1(SKL(1, 1, 1), SKL(1, 1, 2));
end
```

Algorithm 4.18 Algorithm for the computation of the integrand necessary to build the force vector.

```
function F = forceIntegrandBsplines...                                  1
    (n, p, Xi, m, q, Eta, P, xi, eta, ki, li, f)                        2
% Preallocation.                                                       3
F(length(xi)) = 0;                                                      4
for l = 1:length(xi)                                                    5
    % Determination of the derivatives.                                 6
    [SKL, eXi, eEta, spanxi, spaneta] =...                              7
        bsplineSurfDerivs(n, p, Xi, m, q, Eta, P, xi(l), eta, 1);       8
    % Definition of the jacobian matrix.                                9
    DxDxi = [SKL(2, 1, 1), SKL(1, 2, 1); SKL(2, 1, 2), SKL(1, 2, 2)];   10
    % Evaluate the Jacobian.                                            11
    Jx = det(DxDxi);                                                    12
    dki = ki-spanxi+p+1;                                                13
    dli = li-spaneta+q+1;                                               14
    if dki>0 && dki<=p+1 && dli>0 && dli<=q+1                            15
        Nip = eXi(1, dki).*eEta(1, dli);                                16
    else Nip = 0; end;                                                  17
    % Integrand.                                                        18
    F(l) = Jx.*f(SKL(1, 1, 1), SKL(1, 1, 2)).*Nip;                      19
end                                                                     20
```

Algorithm 4.19 Computation of the term of the force vector taking into account the Neumann conditions applied to the boundaries.

```
function F = forceIntegralBsplines02...                                  1
    (n, p, Xi, m, q, Eta, P, ki, li, f, g_N, a_1, pw, N)                 2
F = 0;                                                                   3
% According to the properties of the B-spline basis functions, each     4
% function is nonzero only in a specified interval. This property is useful 5
% in order to reduce the domain of computation of the integral.         6
% The integral is divided over the single elements as along the edges the 7
% degree of continuity is not known: it could lead to a wrong computation 8
% of the integral.                                                      9
for xi_i = ki+1:ki+p+1                                                   10
    for eta_i = li+1:li+q+1                                              11
        F = F+...                                                        12
            dblquad(@(x, y)(forceIntegrandBsplines...                    13
            (n, p, Xi, m, q, Eta, P, x, y, ki, li, f)),...               14
            Xi(xi_i), Xi(xi_i+1),...                                     15
            Eta(eta_i), Eta(eta_i+1), 1e-5);                             16
    end                                                                  17
end                                                                      18
% N stores the definition of which edges of the parametric domain has to be 19
% considered part of the Neumann boundary.                              20
for i = 1:length(N)                                                      21
    if N(i) == 0, continue; end;                                         22
    % Determine the domain of the line integral to compute.              23
    switch i                                                             24
        case 1                                                           25
            if li~=0, continue; end;                                     26
            A = [Xi(1), Eta(1)]; B = [Xi(end), Eta(1)];                  27
            domain(1) = Xi(ki+1);                                        28
            domain(2) = Xi(ki+p+2);                                      29
        case 2                                                           30
            if ki~=n, continue; end;                                     31
            A = [Xi(end), Eta(1)]; B = [Xi(end), Eta(end)];              32
            domain(1) = Eta(li+1);                                       33
            domain(2) = Eta(li+q+2);                                     34
        case 3                                                           35
            if li~=m, continue; end;                                     36
            A = [Xi(1), Eta(end)]; B = [Xi(end), Eta(end)];              37
            domain(1) = Xi(ki+1);                                        38
            domain(2) = Xi(ki+p+2);                                      39
        case 4                                                           40
            if ki~=0, continue; end;                                     41
            A = [Xi(1), Eta(1)]; B = [Xi(1), Eta(end)];                  42
            domain(1) = Eta(li+1);                                       43
            domain(2) = Eta(li+q+2);                                     44
    end                                                                  45
    % Computation of the integral.                                       46
    F = F+quad(@(x)(neumannIntegrand(n, p, Xi, m, q, Eta,...             47
                P, ki, li, a_1, g_N, x, A, B)),...                       48
                domain(1), domain(2));                                   49
end                                                                      50
```

Algorithm 4.20 Integrand of the Neumann term.

```
function N = neumannIntegrand(n, p, Xi, m, q, Eta, P, ki, li,...
    a_1, g_N, t, A, B)
N(length(t)) = 0;
for i = 1:length(t)
    % Definition of the map.
    chi = (1-t(i)).*A + t(i).*B;
    % Determination of the derivatives.
    [SKL, eXi, eEta, spanxi, spaneta] =...
        bsplineSurfDerivs(n, p, Xi, m, q, Eta, P, chi(1), chi(2), 1);
    % Definition of the jacobian matrix.
    DxDxi = [SKL(2, 1, 1), SKL(1, 2, 1); SKL(2, 1, 2), SKL(1, 2, 2)];
    dki = ki-spanxi+p+1;
    dli = li-spaneta+q+1;
    if dki>0 && dki<=p+1 && dli>0 && dli<=q+1
        Nip = eXi(1, dki).*eEta(1, dli);
    else
        Nip = 0;
    end
    N(i) = a_1(SKL(1, 1, 1), SKL(1, 1, 2)).*...
        g_N(SKL(1, 1, 1), SKL(1, 1, 2)).*...
        Nip.*norm(DxDxi*(-A+B)'./norm(-A+B));
end
```

Chapter 5

Numerical examples

5.1 Thermal conduction

Let $u(\boldsymbol{x})$ be the temperature field, $f(\boldsymbol{x})$ the heat supply per unit volume and $\kappa(\boldsymbol{x})$ the conductivity matrix. The steady state thermal conduction problem is the problem of finding the temperature field $u(\boldsymbol{x})$ given that

$$\begin{cases} \nabla(\kappa(\boldsymbol{x})\nabla u(\boldsymbol{x})) = f(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Omega, \text{ given } f : \Omega \to \mathbb{R},\ u : \bar{\Omega} \to \mathbb{R} \\ u(\boldsymbol{x}) = g_D(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Gamma_D,\ g_D : \Gamma_D \to \mathbb{R} \\ \kappa(\boldsymbol{x}) \dfrac{\partial u}{\partial \boldsymbol{\nu}}(\boldsymbol{x}) = g_N(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Gamma_N,\ g_N : \Gamma_N \to \mathbb{R} \end{cases}.$$

The application of the method introduced in 1.4-1.4.4 leads to the problem of finding the function $u(\boldsymbol{x})$ such that

$$\begin{cases} a(\upsilon, \varphi) = l(\varphi), & \forall \varphi \in H_0^1(\Omega),\ \upsilon \in H_0^1(\Omega) \\ a(\upsilon, \varphi) = \iint_\Omega \kappa \nabla \upsilon \nabla \varphi d\boldsymbol{x}, & \upsilon \in H_0^1(\Omega),\ \forall \varphi \in H_0^1(\Omega) \\ l(\varphi) = \iint_\Omega (f\varphi - \kappa \nabla \gamma \nabla \varphi)\, d\boldsymbol{x} - \int_{\Gamma_D} \kappa g_N \varphi d\boldsymbol{\Gamma}, & \forall \varphi \in H_0^1(\Omega) \\ \upsilon(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \Gamma_D \\ \gamma(\boldsymbol{x}) = g_N(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Gamma_D,\ g_D : \Gamma_D \to \mathbb{R} \\ \kappa(\boldsymbol{x}) \dfrac{\partial(\upsilon + \gamma)}{\partial \boldsymbol{\nu}}(\boldsymbol{x}) = g_N(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Gamma_N,\ g_N : \Gamma_N \to \mathbb{R} \\ u(\boldsymbol{x}) = \upsilon(\boldsymbol{x}) + \gamma(\boldsymbol{x}), & \forall \boldsymbol{x} \in \Omega \end{cases}.$$

5.1.1 FEM solution on square plate

Let's consider some different problems on a square plate. The first problem (5.1) is a thermal problem with conductivity matrix $\kappa = \mathbf{I}_{\text{size}(\kappa)}$, domain $\Omega_S = (0,1)^2$, $g_D = 0$ and $f = 1$:

$$\begin{cases} \nabla(\nabla u(\boldsymbol{x})) = 1, & \forall \boldsymbol{x} \in \Omega_S, \text{ given } u : \bar{\Omega}_S \to \mathbb{R} \\ u(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \partial\Omega_S \end{cases}. \tag{5.1}$$

The first process to begin when solving this problem with FEM, after the derivation of the weak formulation and the application of the Galerkin method, is to

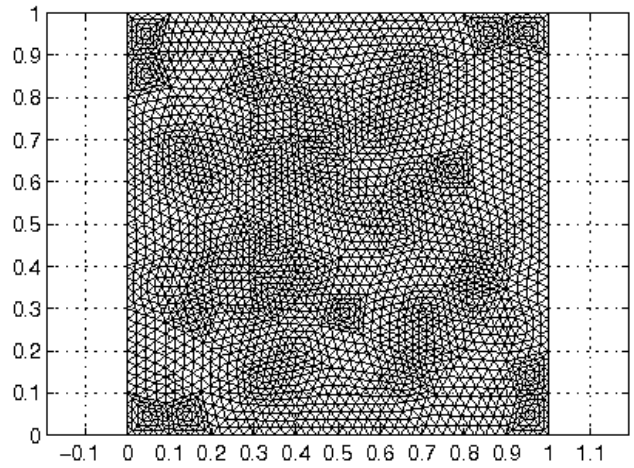

Figure 5.1: Triangular mesh on the rectangular plate for the solution of a simple thermal problem.

create some kind of mesh on the domain of interest, which is described by a CAD model, using splines for instance. Let's consider a simple object at the beginning: a square plate. A possible mesh is given in Figure 5.1. The creation of the stiffness matrix and of the force vector and the solution of the linear system leads to a solution exact at the nodes (see Figure 5.2).

The second problem (5.7) is a thermal problem with conductivity matrix $\kappa = \mathbf{I}_{\text{size}(\kappa)}$, domain $\Omega_S = (0,1)^2$ and $f = 1$:

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_S, \text{ given } u : \bar{\Omega}_S \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,S} = \left\{ \boldsymbol{x} \in \Omega_S | \boldsymbol{x} = [0,y]^T \wedge \boldsymbol{x} = [1,y] \right\} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_S \backslash \Gamma_{D,S} \end{cases}. \tag{5.2}$$

The third problem (5.3) is a thermal problem with conductivity matrix $\kappa = \mathbf{I}_{\text{size}(\kappa)}$, domain $\Omega_S = (0,1)^2$ and $f = x$:

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 20x, & \forall \boldsymbol{x} \in \Omega_S, \text{ given } u : \bar{\Omega}_S \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,S} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_S \backslash \Gamma_{D,S} \end{cases}. \tag{5.3}$$

The fourth problem is:

$$\begin{cases} \nabla\left(2x\mathbf{I}_2 \nabla u\left(\boldsymbol{x}\right)\right) = 20x, & \forall \boldsymbol{x} \in \Omega_S, \text{ given } u : \bar{\Omega}_S \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,S} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_S \backslash \Gamma_{D,S} \end{cases}. \tag{5.4}$$

The solutions to the problems (5.2), (5.3) and (5.4) can be found in Figure 5.3.

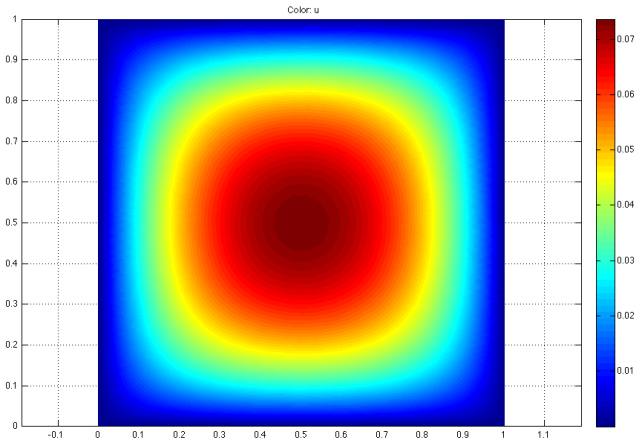

Figure 5.2: Approximated solution of a thermal problem with FEM on the square plate using a mesh with 2705 nodes.

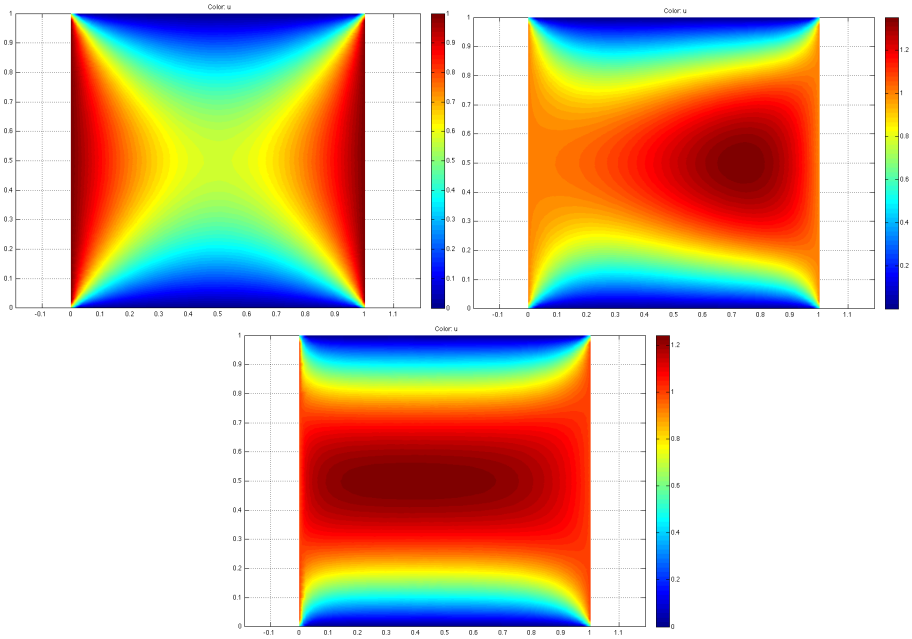

Figure 5.3: Approximated solutions of problems (5.2), (5.3) and (5.4) found using FEM with a mesh with 2705 nodes.

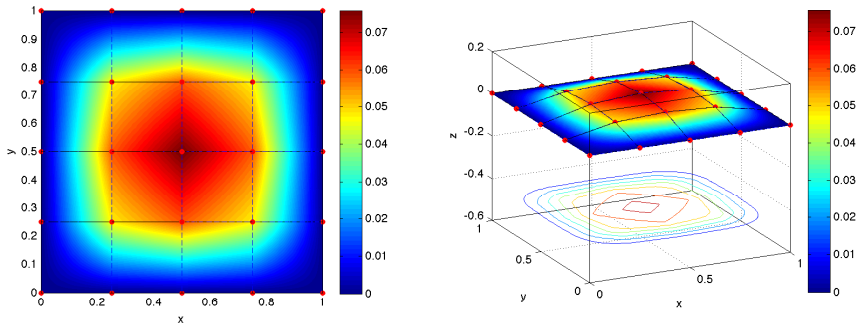

Figure 5.4: Representations of the approximated solutions of the problem (5.1) calculated using IGA, with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

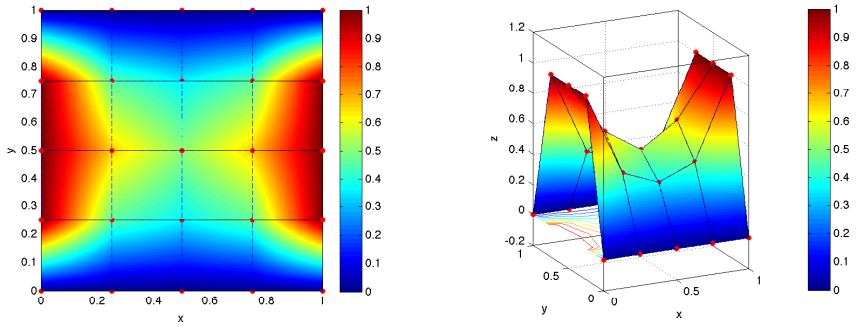

Figure 5.5: Representations of the approximated solutions of the problem (5.2) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

5.1.2 IGA solution on square plate

The results of the analysis of the problems (5.1) to (5.4) with IGA can be seen in Figures from 5.4 to 5.7.

A comparison of FEM and IGA solutions using the same number of nodes is reported in Figure 5.8.

5.1.3 FEM solution on a plate with hole

Now, it is interesting to consider some more complex geometries. Let's take, for instance, an approximation of the plate with a hole in the corner described using a B-spline surface, like the one represented in Figure 3.18. It is no more possible to keep the exact geometry when trying to analyze the domain with FEM. In this case, in fact, the "circular" hole needs to be approximated with a polygonal shape (linear elements are used in this case). The refinement of the mesh can represent the domain with higher precision, and this is useful in this case, as it can be seen in Figure 5.9. The domain Ω_P is described using a B-spline surface $\boldsymbol{P}(\xi, \eta)$ where the parametric space is always the unit square

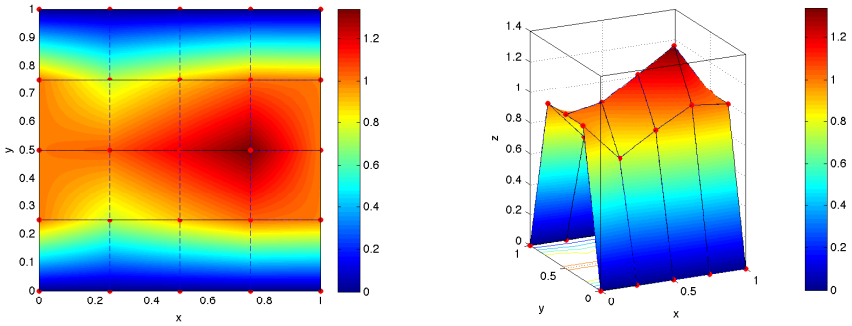

Figure 5.6: Representations of the approximated solutions of the problem (5.3) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

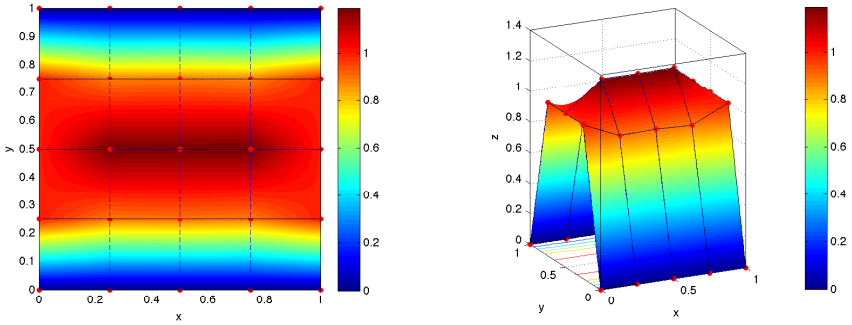

Figure 5.7: Representations of the approximated solutions of the problem (5.4) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

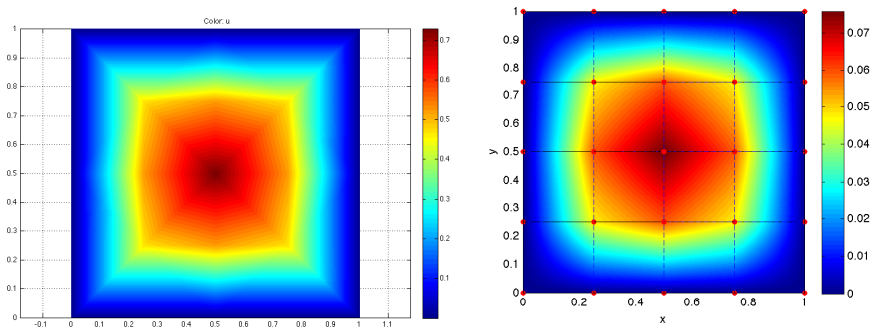

Figure 5.8: Approximated solutions using FEM with 25 nodes on the left, IGA with 25 nodes on the right.

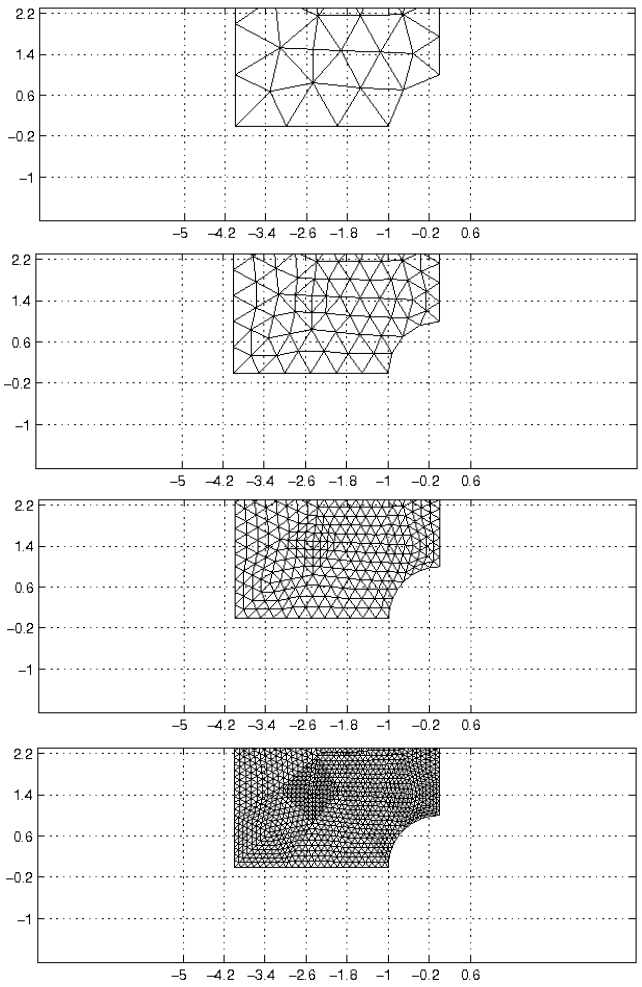

Figure 5.9: Details of meshes for the plate with hole.

$[0,1]^2$, so

$$\Omega_P = \left\{ (x,y) \,|\, (x,y) = \boldsymbol{P}(\xi,\eta),\ \forall (\xi,\eta) \in [0,1]^2 \right\}.$$

We consider again four problems:

$$\begin{cases} \nabla(\nabla u(\boldsymbol{x})) = 1, & \forall \boldsymbol{x} \in \Omega_P, \text{ given } u : \bar{\Omega}_P \to \mathbb{R} \\ u(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \partial\Omega_P \end{cases}, \tag{5.5}$$

$$\begin{cases} \nabla(\nabla u(\boldsymbol{x})) = 1, & \forall \boldsymbol{x} \in \Omega_P, \text{ given } u : \bar{\Omega}_P \to \mathbb{R} \\ u(\boldsymbol{x}) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,P} \\ u(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \Omega_P \backslash \Gamma_{D,P} \end{cases}, \tag{5.6}$$

$$\begin{cases} \nabla(\nabla u(\boldsymbol{x})) = x, & \forall \boldsymbol{x} \in \Omega_P, \text{ given } u : \bar{\Omega}_P \to \mathbb{R} \\ u(\boldsymbol{x}) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,P} \\ u(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \Omega_P \backslash \Gamma_{D,P} \end{cases}, \tag{5.7}$$

$$\begin{cases} \nabla(x\mathbf{I}_2 \nabla u(\boldsymbol{x})) = 1, & \forall \boldsymbol{x} \in \Omega_P, \text{ given } u : \bar{\Omega}_P \to \mathbb{R} \\ u(\boldsymbol{x}) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,P} \\ u(\boldsymbol{x}) = 0, & \forall \boldsymbol{x} \in \Omega_P \backslash \Gamma_{D,P} \end{cases}, \tag{5.8}$$

where

$$\Gamma_{D,P} = \left\{ \boldsymbol{x} \in \Omega_P | \boldsymbol{x} = [-4, y]^T \wedge \boldsymbol{x} = [x, 4] \right\}.$$

The solutions to the problems (5.5), (5.6), (5.7) and (5.8) can be found in Figure 5.10.

5.1.4 IGA solution on a plate with hole

When the plate with hole is described using B-spline functions and IGA is applied, there is no need to approximate the boundary. The effect of h-refinement can be seen in Figures 5.20 and 5.21, and it can be seen that in every mesh, including the coarsest, the domain is represented exactly as it was provided. The solutions to the problems (5.5), (5.6), (5.7) and (5.8) can be found in Figures from 5.11 to 5.14.

A comparison of the approximated solutions found using FEM and IGA on the square plate with hole can be found in Figure 5.15. In this case, the advantage of using IGA over FEM can be clearly seen.

5.1.5 IGA on Ω_P with optimized Gauss quadrature

The results of the analysis of the problems (5.5) to (5.8) with IGA can be seen in Figures from 5.16 to 5.19.

5.1.6 Effects of h-refinement

Algorithms 4.3 and 4.4, which performs the knot insertion, can be used to refine a mesh on an object. This is equivalent to the h-refinement in FEM, so that it can be used to obtain a more precise solution. An example of h-refinement with B-splines functions can be found in Figure 5.20: a uniform mesh is uniformly refined in order to obtain a more precise approximation of the exact solution.

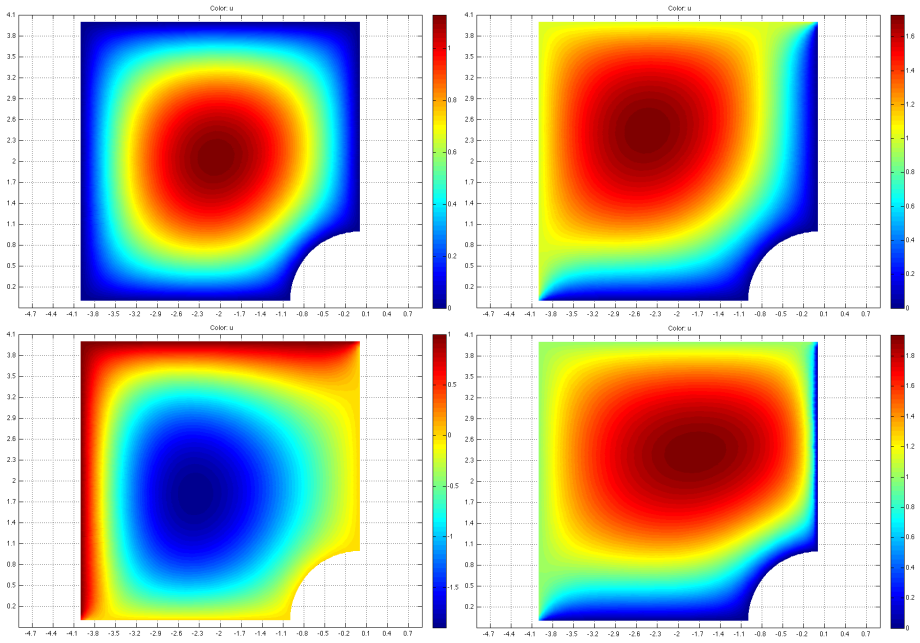

Figure 5.10: Approximated solutions of the problems (5.5) on the top left, (5.6) on the top right, (5.7) on the bottom left and (5.8) on the bottom right, using linear elements in FEM analysis with a mesh with 2609 nodes.

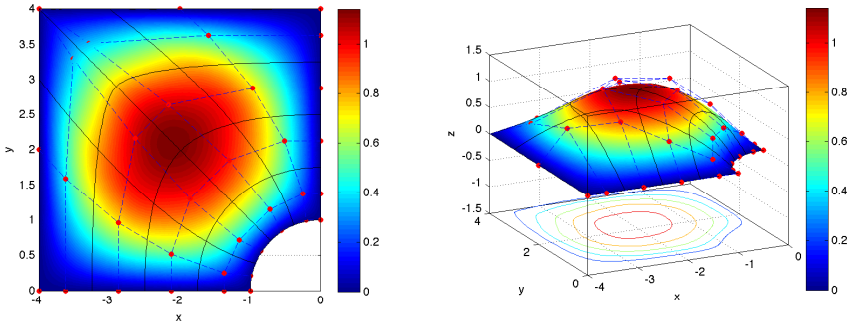

Figure 5.11: Representations of the approximated solution of the problem (5.5) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

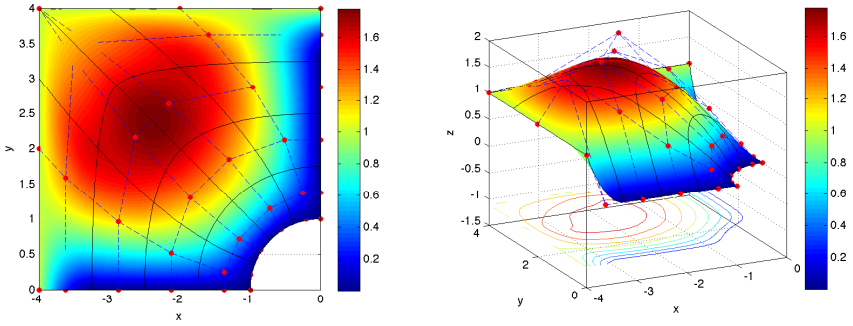

Figure 5.12: Representations of the approximated solution of the problem (5.6) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

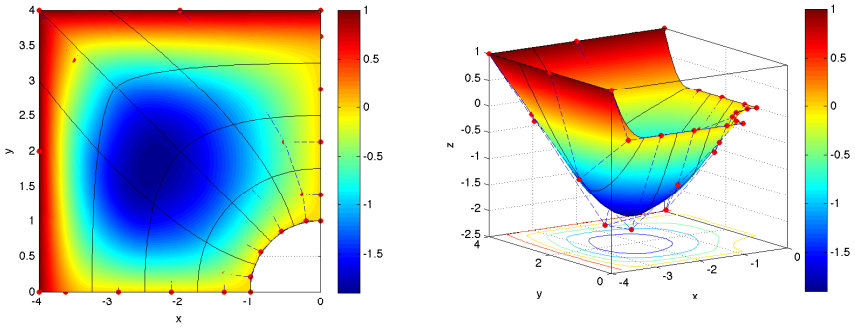

Figure 5.13: Representations of the approximated solution of the problem (5.7) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

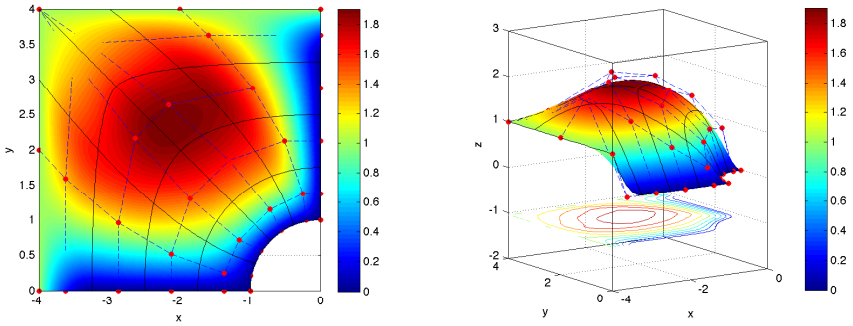

Figure 5.14: Representations of the approximated solution of the problem (5.8) calculated using IGA with integration using adaptive recursive Simpson's rule, with a mesh of 4 elements in each direction.

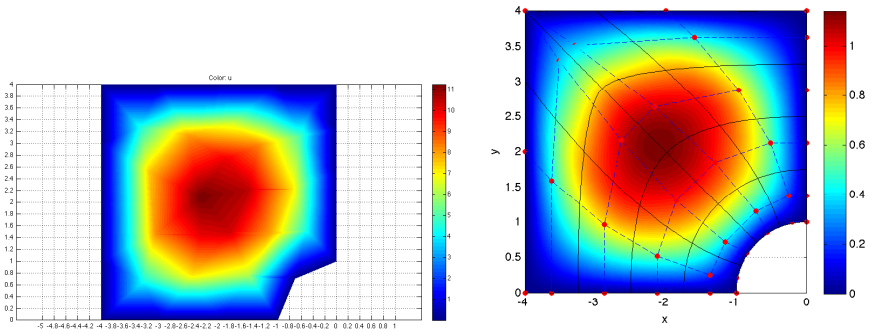

Figure 5.15: Approximated solutions using FEM with 36 nodes on the left, IGA with 35 nodes on the right.

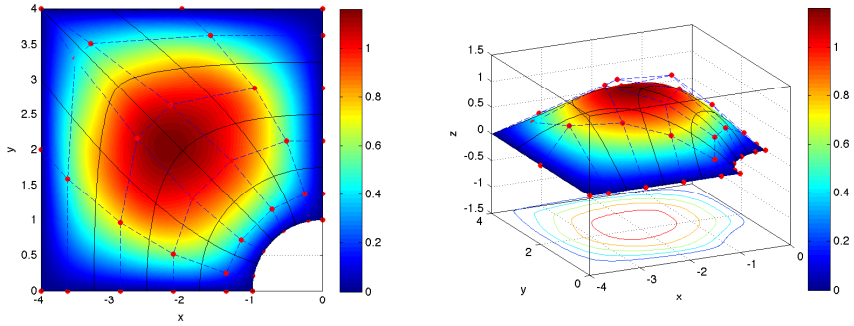

Figure 5.16: Representations of the approximated solution of the problem (5.5) calculated using IGA with optimized Gauss two-dimensional integration, with a mesh of 4 elements in each direction.

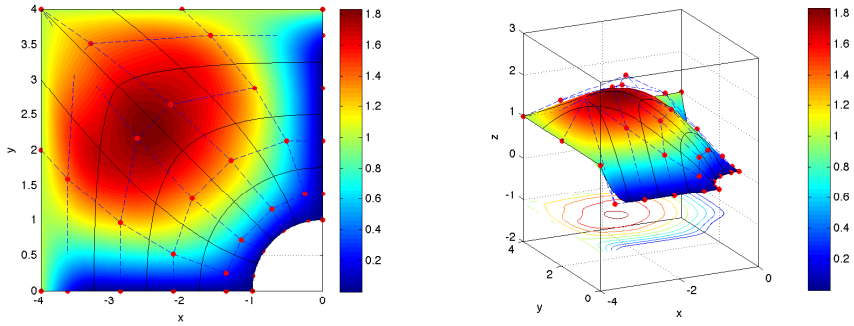

Figure 5.17: Representations of the approximated solution of the problem (5.6) calculated using IGA with optimized Gauss two-dimensional integration, with a mesh of 4 elements in each direction.

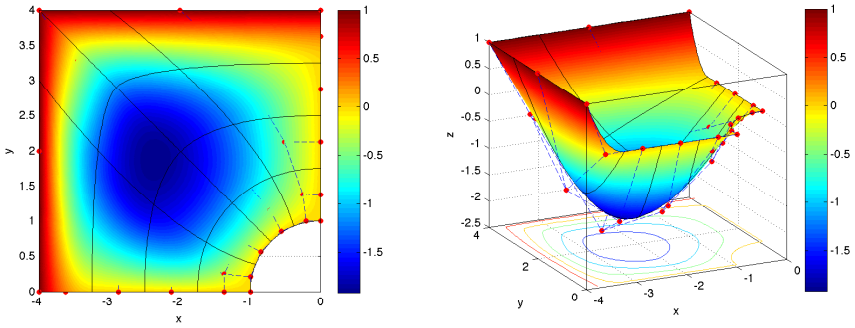

Figure 5.18: Representations of the approximated solution of the problem (5.7) calculated using IGA with optimized Gauss two-dimensional integration, with a mesh of 4 elements in each direction.

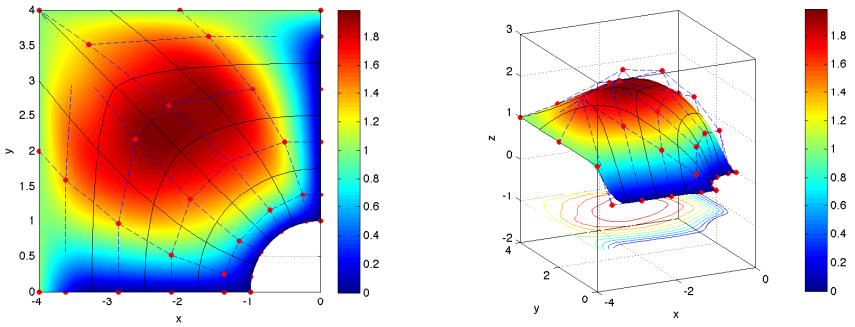

Figure 5.19: Representations of the approximated solution of the problem (5.8) calculated using IGA with optimized Gauss two-dimensional integration, with a mesh of 4 elements in each direction.

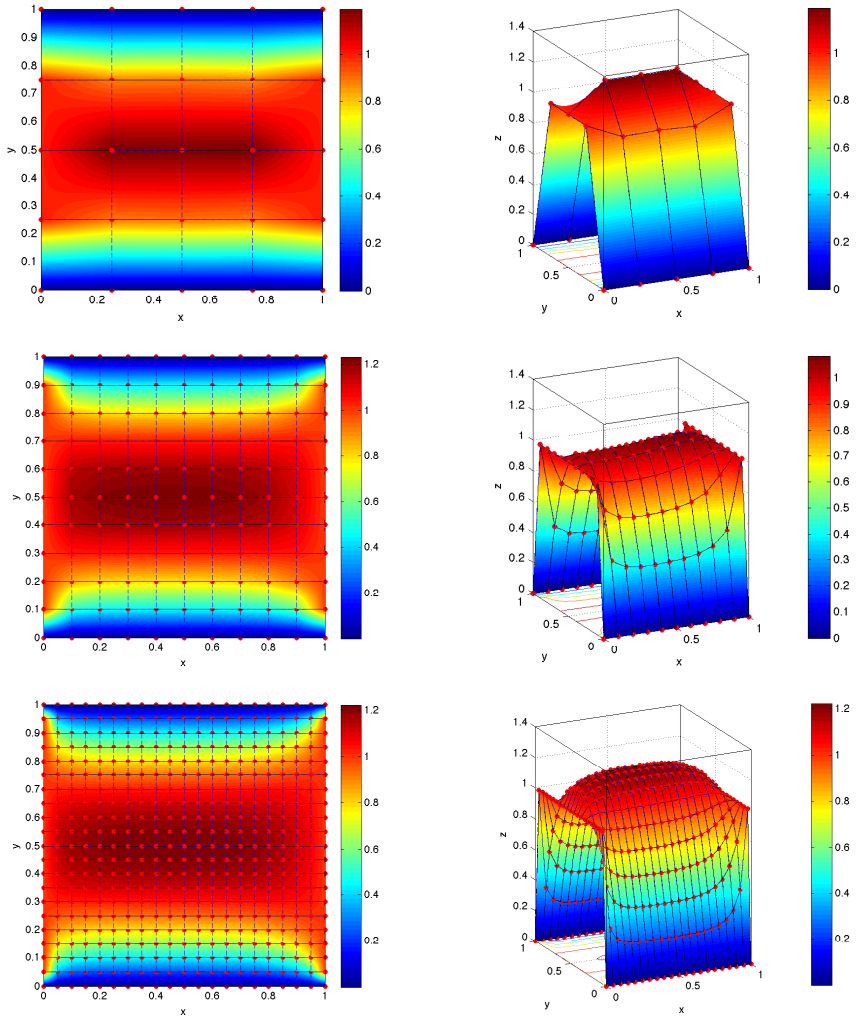

Figure 5.20: Process of h-refinement on Ω_S for the problem (5.4).

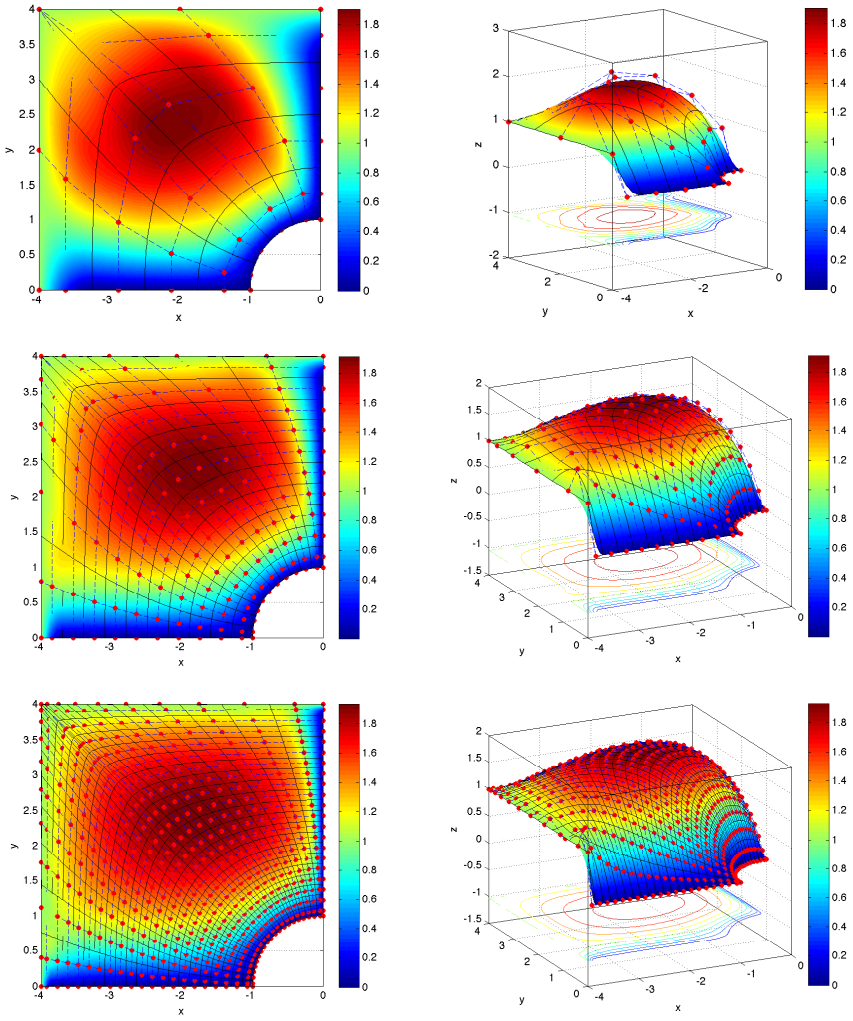

Figure 5.21: Process of h-refinement on Ω_P for the problem (5.8).

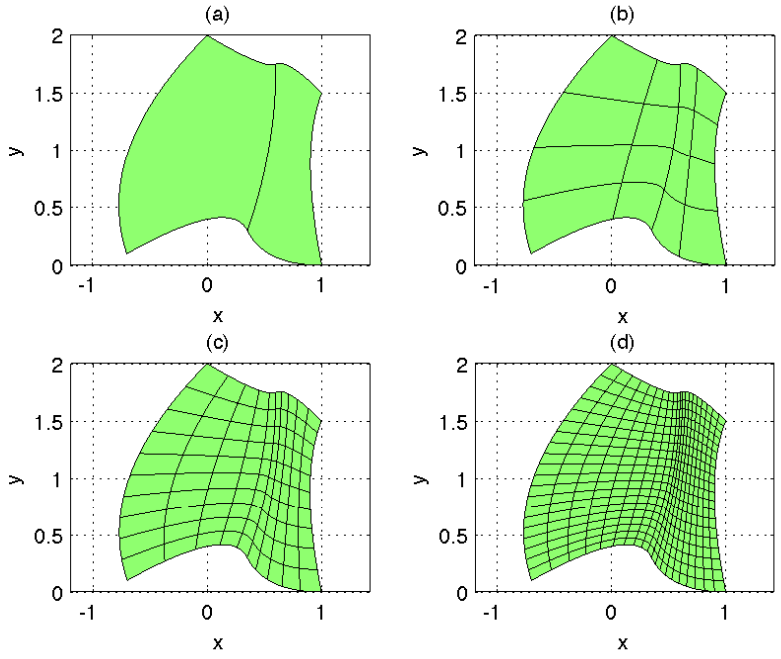

Figure 5.22: h-refinement of domain Ω_1.

5.1.7 IGA solution on a more complex domain

By using IGA, it is possible to analyze even more complex geometries. Let's consider then problem (5.9)

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_1, \text{ given } u : \bar{\Omega}_1 \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_1 \end{cases}, \tag{5.9}$$

where Ω_1 is represented in Figure 5.22 in its coarsest mesh and its refinements.

In this case, IGA provides the solution (mesh of Figure 5.22c is considered) depicted in Figure 5.23.

Another interesting problem to test on this complex domain is the one reported below:

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_1, \text{ given } u : \bar{\Omega}_1 \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_1 \backslash \Gamma_D \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_D = \left\{\boldsymbol{x} | \boldsymbol{x} = \tilde{\boldsymbol{x}}\left(\xi, \eta\right), \eta = 1\right\} \end{cases}. \tag{5.10}$$

The solution is illustrated in Figure 5.24, and it can be seen that the Dirichlet condition is exactly satisfied on the boundary.

IGA can also work on three-dimensional surfaces, consider in fact the case of a problem similar to (5.9) such as:

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_2, \text{ given } u : \bar{\Omega}_2 \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \partial\Omega_2 \end{cases}. \tag{5.11}$$

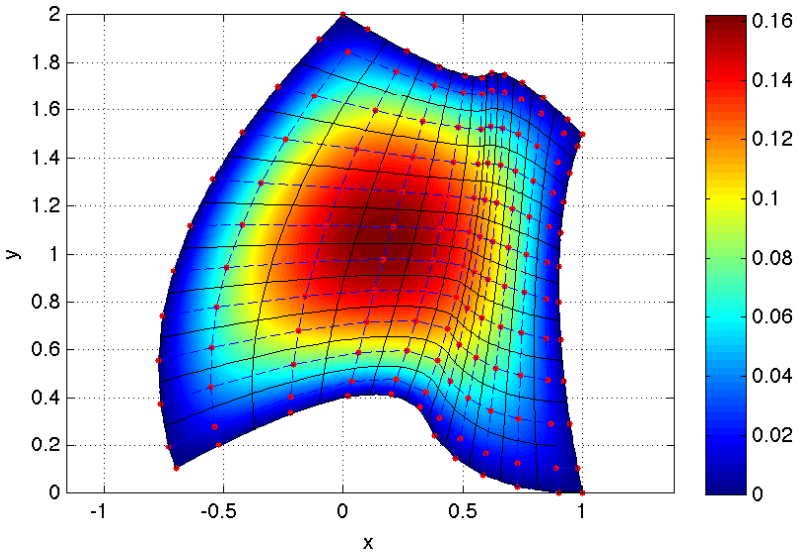

Figure 5.23: Solution of problem (5.11) found by using IGA.

Refinements and the solution over a mesh are reported in Figure 5.26 and 5.25.

5.1.8 Combination of Dirichlet and Neumann boundary conditions

It is possible to introduce Neumann boundary conditions in the problems already proposed. The three problems are:

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_S, \text{ given } u : \bar{\Omega}_S \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \Gamma_{D,0} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,1} \\ u\left(\boldsymbol{x}\right) = 2, & \forall \boldsymbol{x} \in \Gamma_{D,2} \\ \dfrac{\partial u}{\partial \boldsymbol{\nu}} = g_N, & \forall \boldsymbol{x} \in \Gamma_N \end{cases}, \tag{5.12}$$

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_P, \text{ given } u : \bar{\Omega}_P \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0, & \forall \boldsymbol{x} \in \Gamma_{D,0} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,1} \\ u\left(\boldsymbol{x}\right) = 2, & \forall \boldsymbol{x} \in \Gamma_{D,2} \\ \dfrac{\partial u}{\partial \boldsymbol{\nu}} = g_N, & \forall \boldsymbol{x} \in \Gamma_N \end{cases}, \tag{5.13}$$

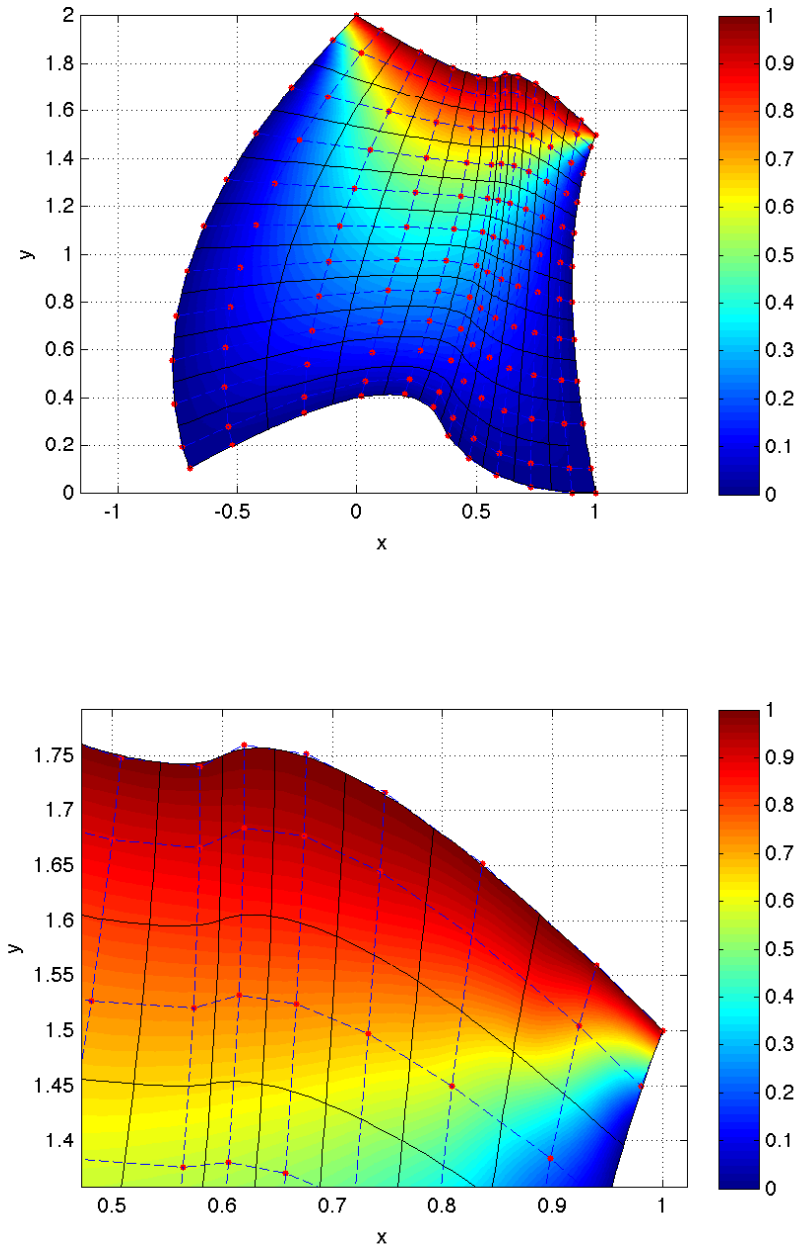

Figure 5.24: Solution of problem (5.10) found by using IGA.

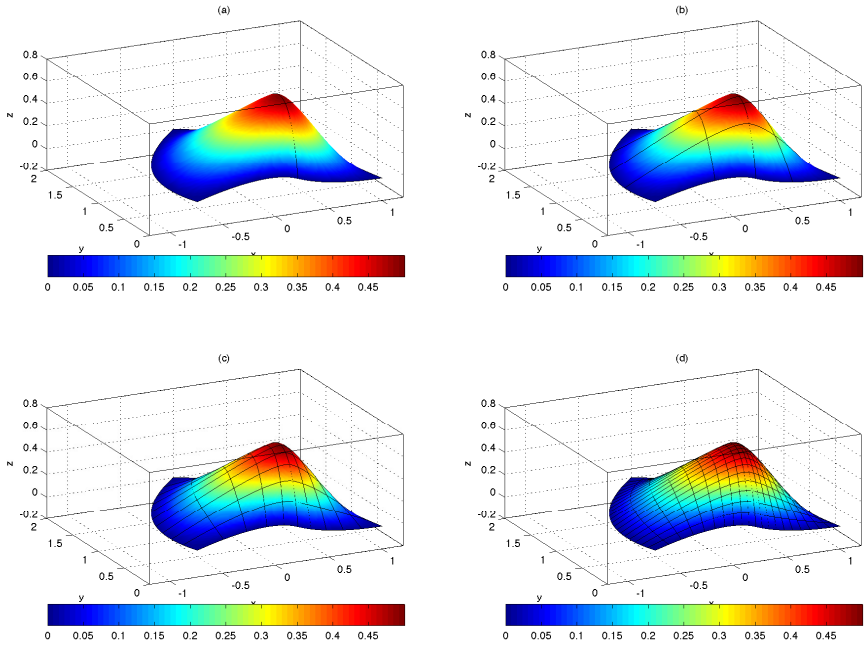

Figure 5.25: h-refinement of domain Ω_2.

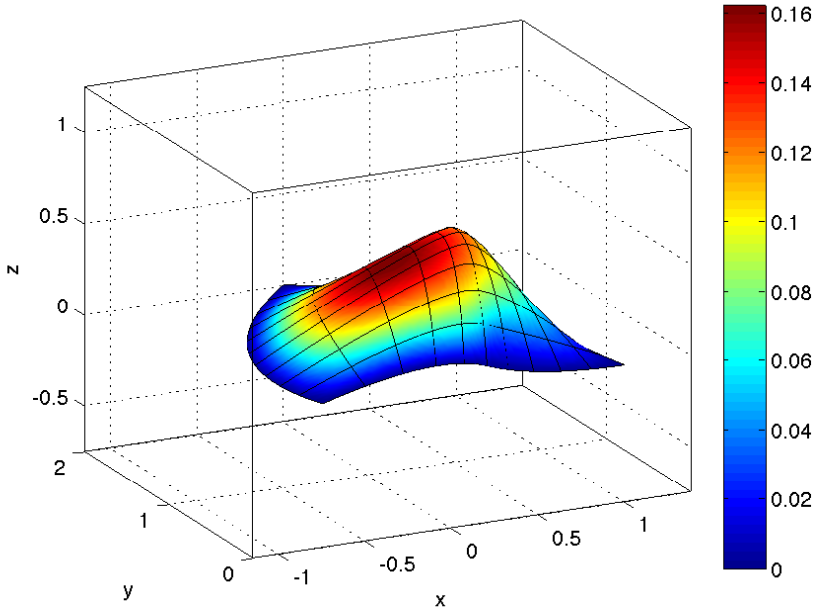

Figure 5.26: Solution of problem (5.11) found by using IGA.

$$\begin{cases} \nabla\left(\nabla u\left(\boldsymbol{x}\right)\right) = 1, & \forall \boldsymbol{x} \in \Omega_1, \text{ given } u : \bar{\Omega}_1 \to \mathbb{R} \\ u\left(\boldsymbol{x}\right) = 0 & \forall \boldsymbol{x} \in \Gamma_{D,0} \\ u\left(\boldsymbol{x}\right) = 1, & \forall \boldsymbol{x} \in \Gamma_{D,1} \\ u\left(\boldsymbol{x}\right) = 2, & \forall \boldsymbol{x} \in \Gamma_{D,2} \\ \dfrac{\partial u}{\partial \boldsymbol{\nu}} = g_N, & \forall \boldsymbol{x} \in \Gamma_N \end{cases}, \tag{5.14}$$

where

$$\Gamma_{D,0} = \left\{\boldsymbol{x} | \tilde{\boldsymbol{x}}\left(\xi, \eta\right) = \boldsymbol{x},\ \xi = 0,\ \eta \in [0,1]\right\},$$
$$\Gamma_{D,1} = \left\{\boldsymbol{x} | \tilde{\boldsymbol{x}}\left(\xi, \eta\right) = \boldsymbol{x},\ \xi = 1,\ \eta \in [0,1]\right\},$$
$$\Gamma_{D,2} = \left\{\boldsymbol{x} | \tilde{\boldsymbol{x}}\left(\xi, \eta\right) = \boldsymbol{x},\ \xi \in [0,1],\ \eta = 1\right\},$$
$$\Gamma_N = \left\{\boldsymbol{x} | \tilde{\boldsymbol{x}}\left(\xi, \eta\right) = \boldsymbol{x},\ \xi \in [0,1],\ \eta = 0\right\}.$$

In the figures from (5.27) to (5.29) the solutions for $h = 0$ can be seen and in the figures from (5.30) to (5.32) $h = 1$ is illustrated. Tests with a nonconstant function for Neumann conditions $g_N\left(\boldsymbol{x}\right) = x$ is also reported in the figure from (5.33) to (5.35).

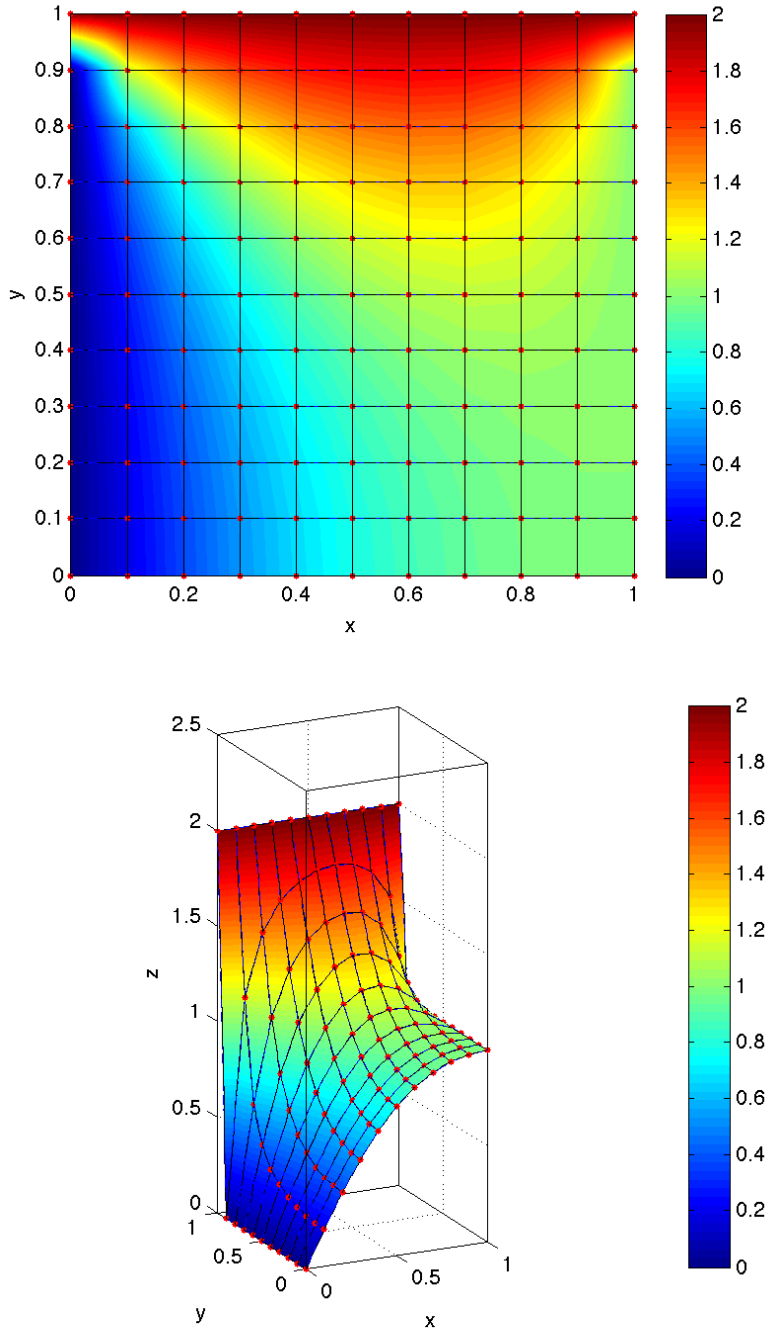

Figure 5.27: Solution of the problem (5.12) on the domain Ω_S with $g_N = 0$.

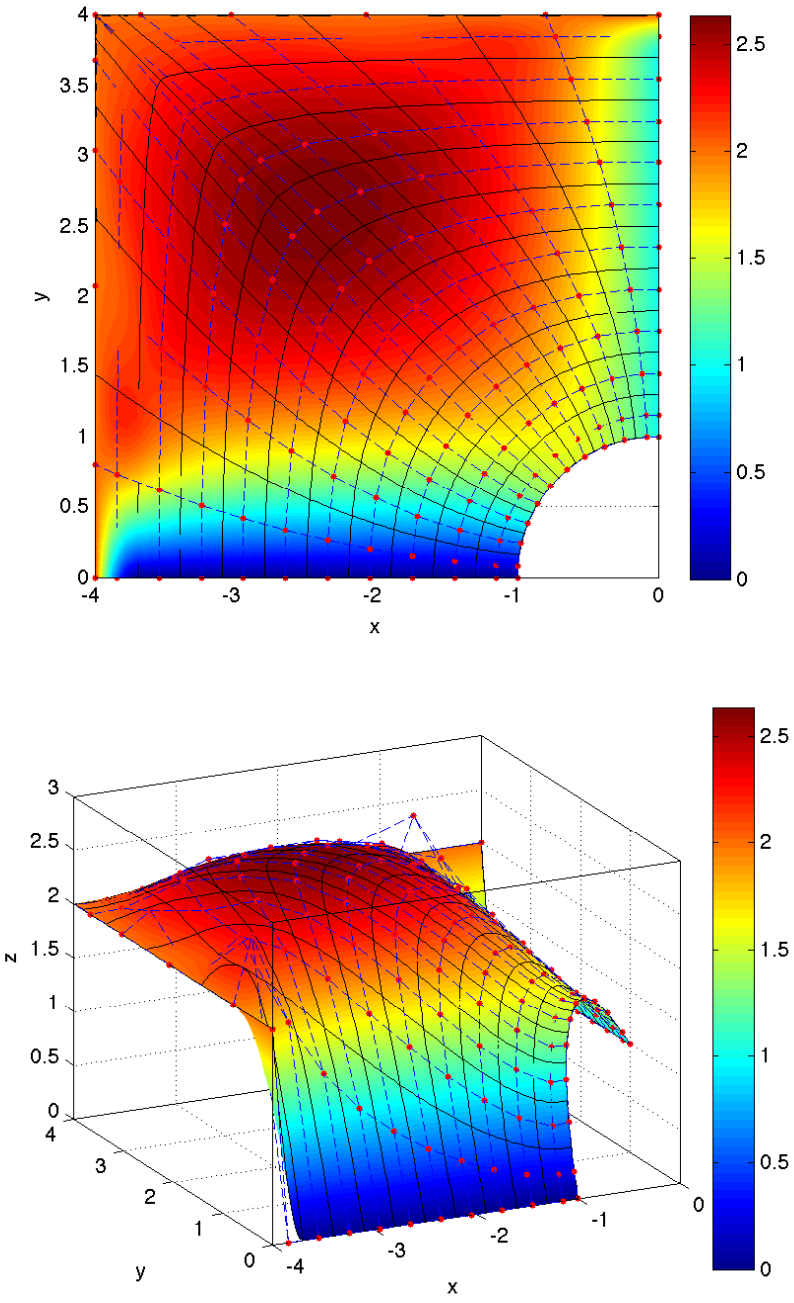

Figure 5.28: Solution of the problem (5.13) on the domain Ω_P with $g_N = 0$.

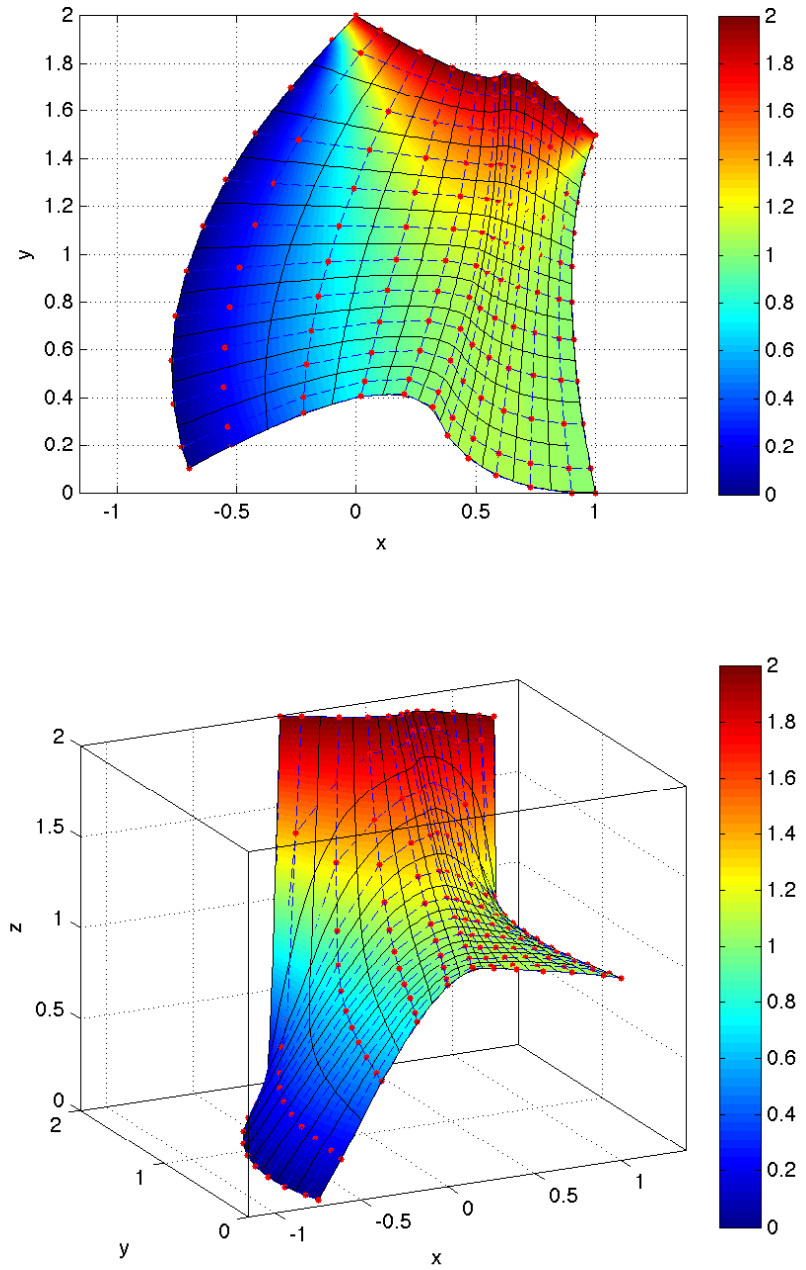

Figure 5.29: Solution of the problem (5.14) on the domain Ω_1 with $g_N = 0$.

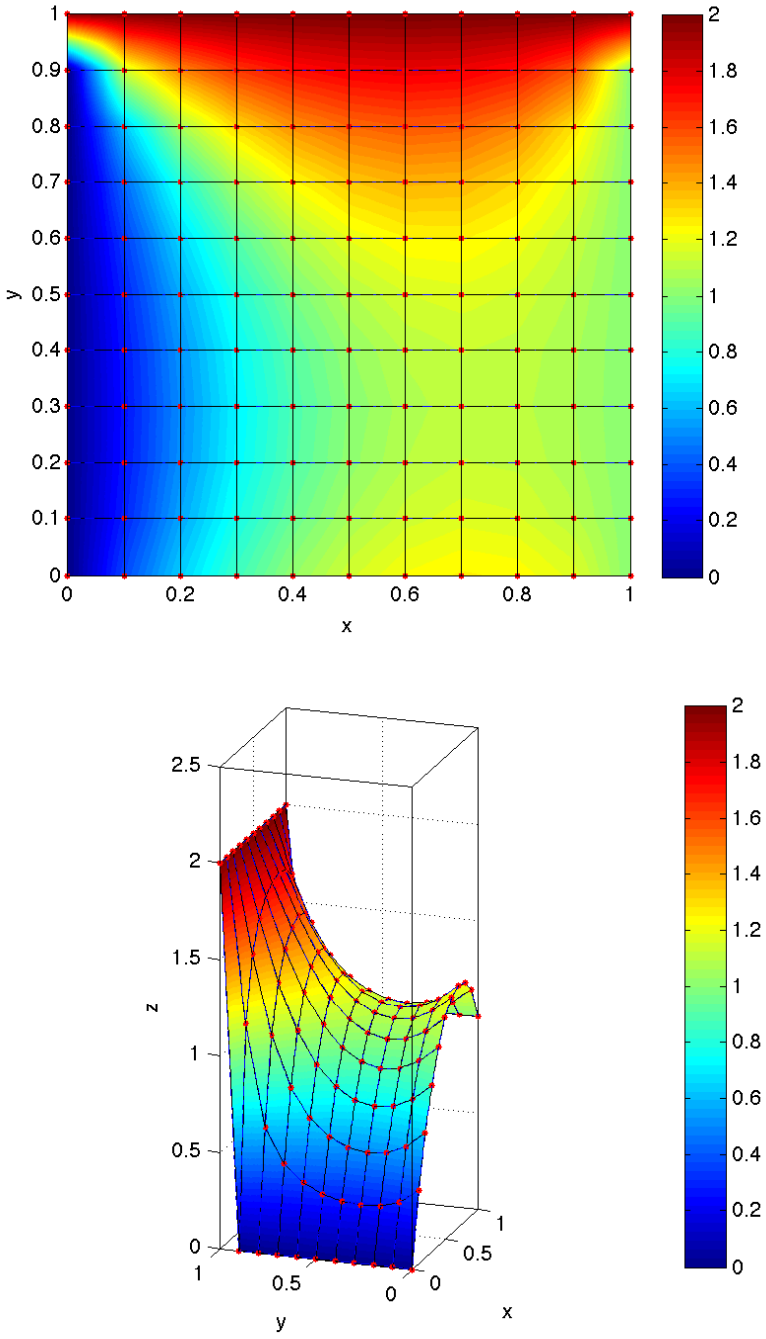

Figure 5.30: Solution of the problem (5.12) on the domain Ω_S with $g_N = 1$.

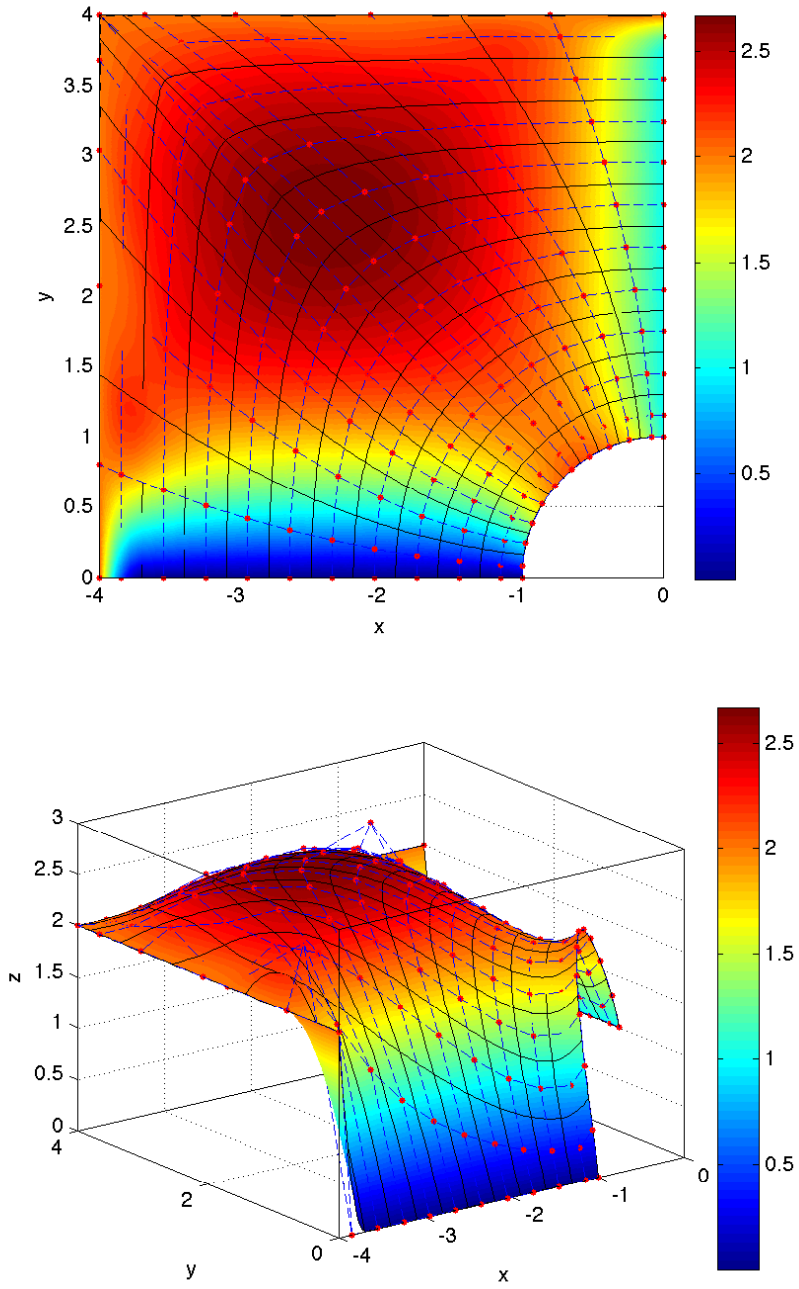

Figure 5.31: Solution of the problem (5.13) on the domain Ω_P with $g_N = 1$.

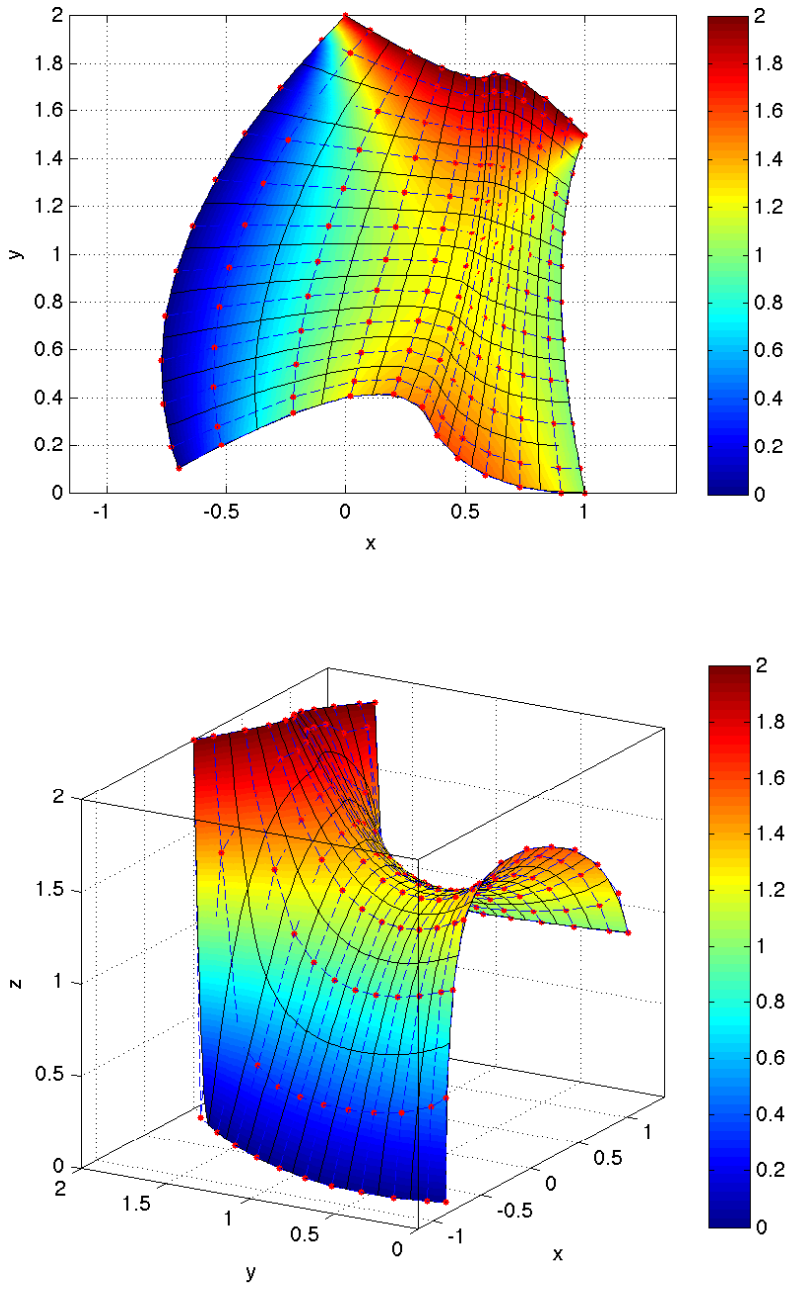

Figure 5.32: Solution of the problem (5.14) on the domain Ω_1 with $g_N = 1$.

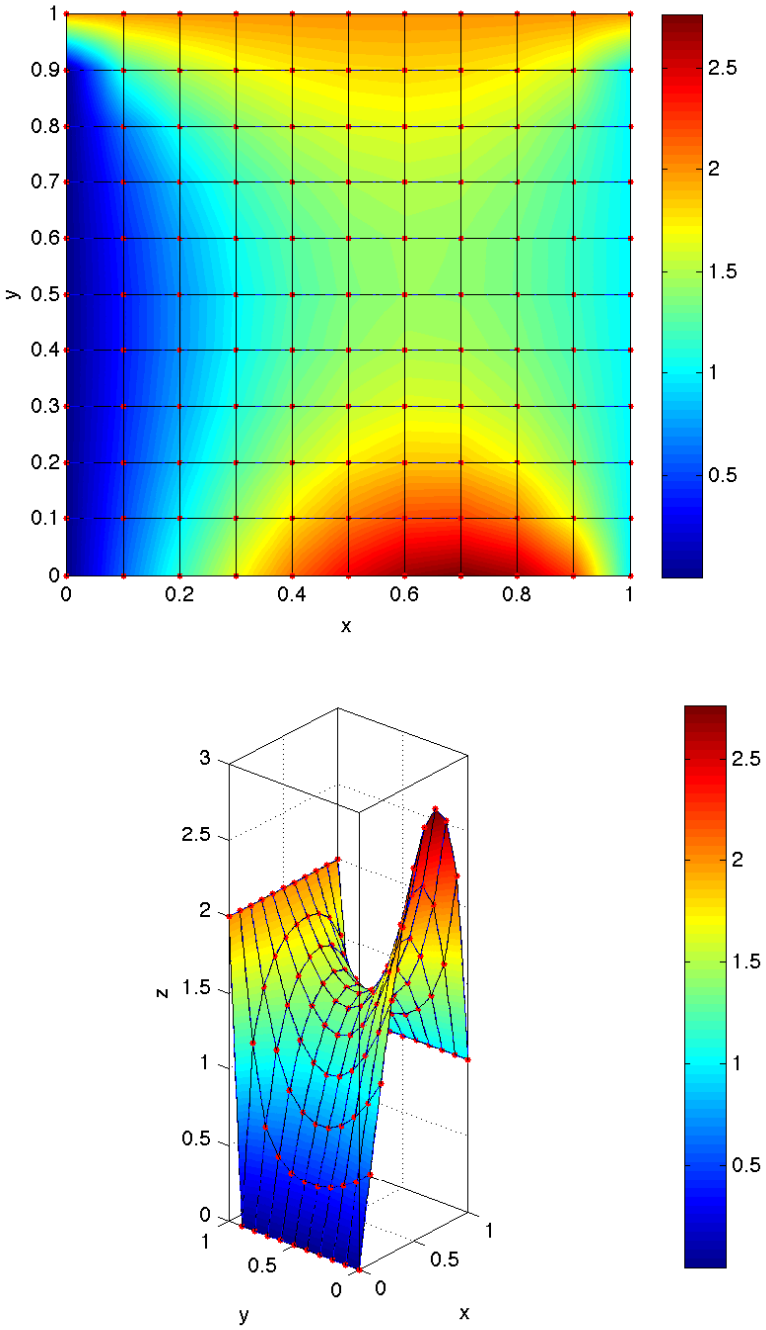

Figure 5.33: Solution of the problem (5.12) on the domain Ω_S with $g_N = 9x$.

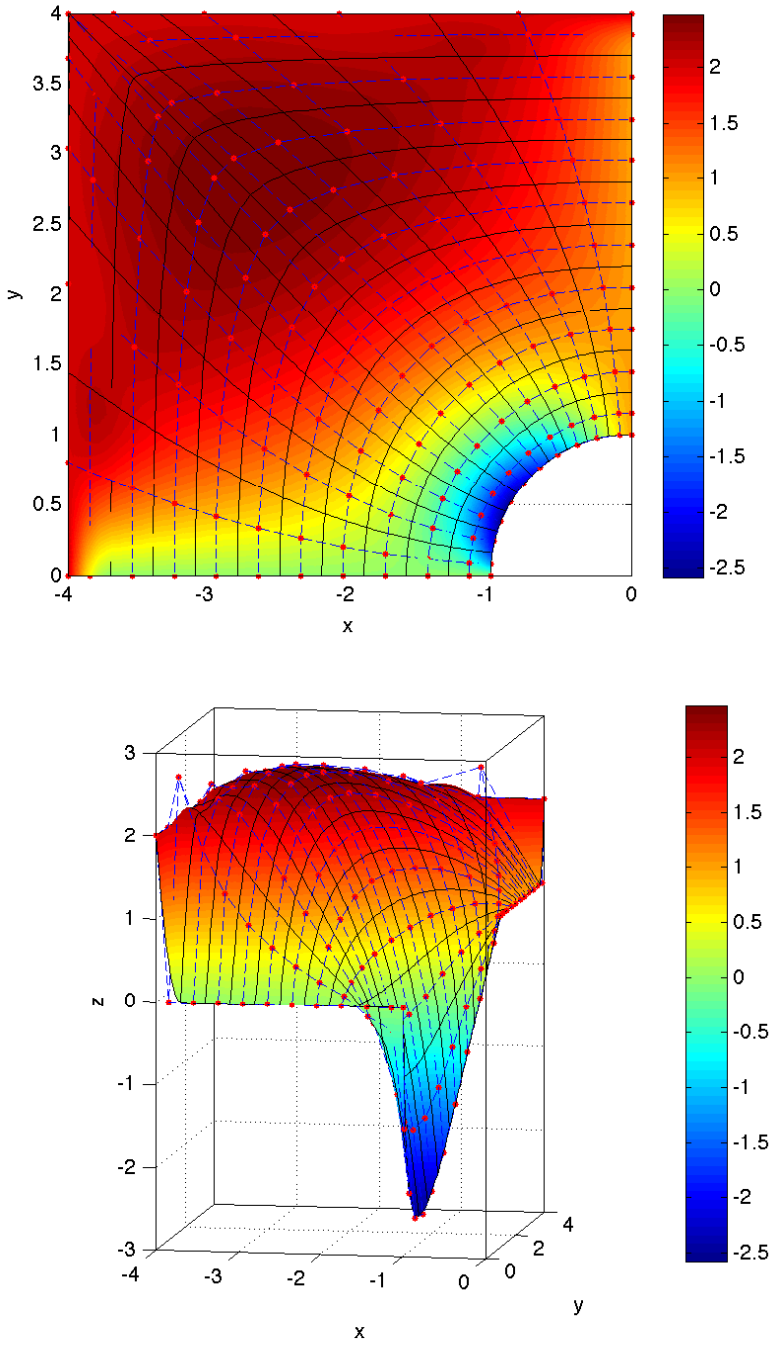

Figure 5.34: Solution of the problem (5.13) on the domain Ω_P with $g_N = 9x$.

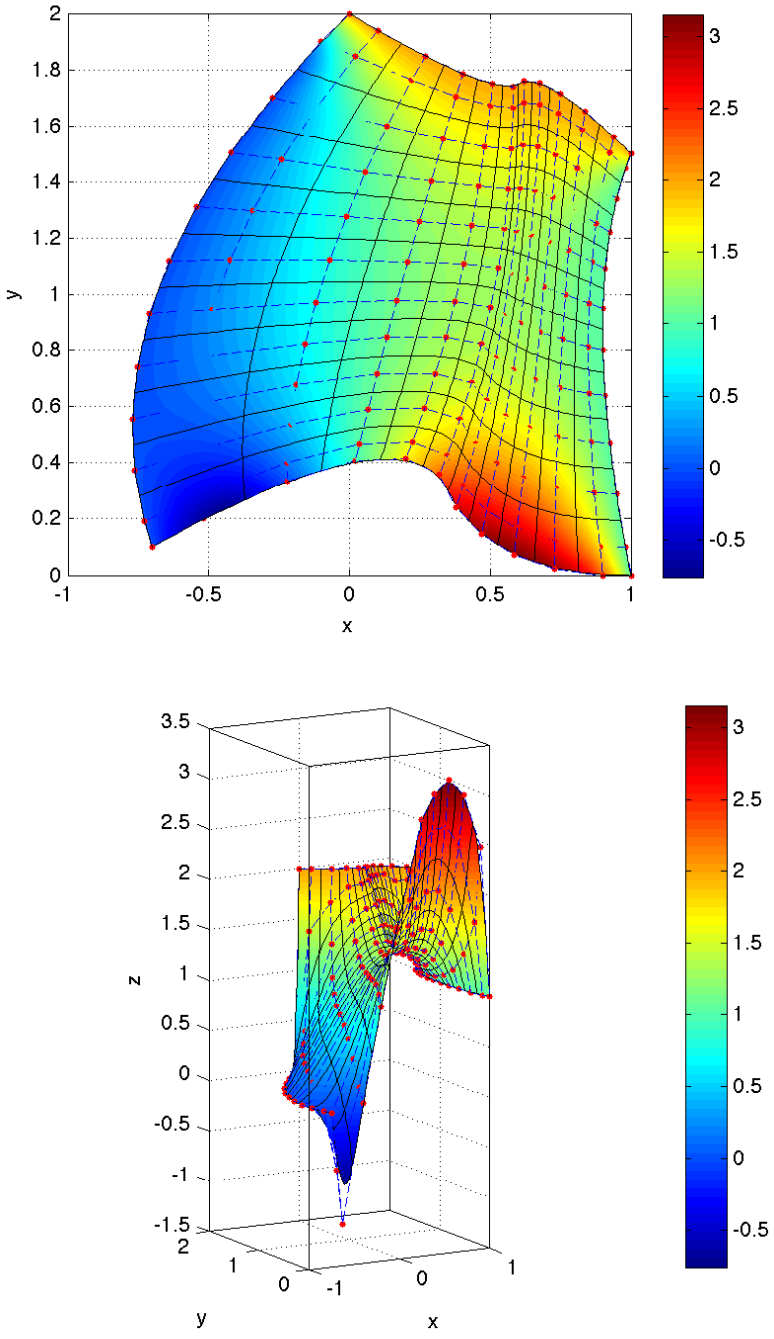

Figure 5.35: Solution of the problem (5.14) on the domain Ω_1 with $g_N = 9x$.

Appendix A

Notes of functional analysis

A.1 Linear spaces

A.1.1 Real and complex linear space

Definition A.1. Let $\mathfrak{B} = \mathbb{R}$ or $\mathfrak{B} = \mathbb{C}$. A nonempty abstract set V endowed with two binary operations $+ : V \times V \to V$ (called addition) and $\cdot : \mathfrak{B} \times V \to V$ (called multiplication by scalar) is (real or complex) linear space if and only if the following ten conditions are satisfied for all $a, b \in \mathfrak{B}$ and $u, v, w \in V$.

1. $v + w \in V$ (closure of V under addition);
2. $u + (v + w) = (u + v) + w$ (associativity of addition in V);
3. there exists a neutral element 0 in V such that for all elements $v \in V$, $v + 0 = v$;
4. for all $v \in V$ there exists an element $w \in V$ such that $v + w = 0$;
5. $v + w = w + v$ (commutativity);
6. $a \cdot v \in V$ (closure of V under multiplication by a scalar);
7. $a \cdot (b \cdot v) = (ab) \cdot v$ (associativity of scalar multiplication);
8. if 1 denotes the multiplicative identity of $\mathfrak{B}$ then $1 \cdot v = v$ (neutrality of 1);
9. $a \cdot (v + w) = a \cdot v + a \cdot w$ (distributivity with respect to addition);
10. $(a + b) \cdot v = a \cdot v + b \cdot v$ (distributivity with respect to scalar addition).

A.1.2 Linear and bilinear forms

Linear forms are linear operators which are commonly used in Finite Element analysis. To define them we define first a linear transformation, then the linear operator and lastly the linear functional (or linear form).

Definition A.2. Let V and W be linear spaces over the same field F. A *linear transformation* is a function $f : V \to W$ such that:

- $f(v+w) = f(v) + f(w)\,,\ \forall v, w \in V$;
- $f(\lambda v) = \lambda f(v)\,,\ \forall v \in V, \lambda \in F$.

Definition A.3. Let f be a linear transformation $f : V \to W$, where V and W are linear spaces over the field F. If $V = W$ then the linear transformation is called *linear operator*.

Definition A.4. Let V be a linear space over the field F. The function $f : V \to F$ is called *linear form*. The space of all linear forms over the space V is called dual space and denoted with V'.

We need to define the dual space as well:

Definition A.5. Let V be a real or complex linear space of dimension n and $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ a basis of V. The basis $\mathcal{V}' = \{v_1', v_2', \ldots, v_n'\}$ of the space V' is said to be the *dual basis* of V if $f_i(v_j) = \delta_{ij},\ \forall 1 \leq i, j \leq n$.

Another important linear form frequently used in Finite Element analysis is the bilinear form.

Definition A.6. Let V, W and Z be linear spaces over the field F. A bilinear map is a function $f : V \times W \to Z$ such that:

- the map $x \mapsto f(x, y)$ from V to Z is a linear $\forall y \in W$;
- the map $y \mapsto f(x, y)$ from W to Z is linear $\forall x \in V$.

Definition A.7. Let V be a linear spaces on a field F. A bilinear form is a bilinear map $f : V \times V \to F$.

Similarly, we can give the definition of a bilinear form:

Definition A.8. A bilinear form on a linear space V over a field V is a bilinear mapping $f : V \times V \to F$, such that:

1. $f(v + v', w) = f(v, w) + f(v', w)\,,\ \forall v, v', w \in V$;
2. $f(w, v + v') = f(w, v) + f(w, v')\,,\ \forall v, v', w \in V$;
3. $f(\lambda v, w) = f(v, \lambda w) = \lambda f(v, w)\,,\ \forall v, w \in V$.

There are some important properties used in the rest of the document for the bilinear form:

Definition A.9. Let V be a real Hilbert space $a : V \times V \to \mathbb{R}$ a bilinear form and $A : V \to V'$ a linear operator related to $a(\cdot, \cdot)$ via

$$(Au)(v) = \langle Au, v \rangle = a(u, v)\,,\ \forall u, v \in V,$$

we say that:

1. a is bounded if there exists a constant C_a such that $|a(u, v)| \leq C_a \|u\| \|v\|$ for all $u, v \in V$;
2. a is V-elliptic if there exists a constant $\tilde{C}_a > 0$ such that $a(u, v) \geq \tilde{C}_a \|v\|_V^2$ for all $v \in V$.

A.2 Normed spaces

A.2.1 Open and closed sets

Some definitions regarding the set theory are necessary.

Definition A.10. Let V be a normed space and $S \subset V$ a subset of V. We say that S is *bounded* in V if there exists a positive constant $c > 0$ such that $\|x\|_V \leq c$ for all $x \in S$.

Definition A.11. Let V be a normed space with the norm $\|\cdot\|_V$, $g \in V$ and $0 < r \in \mathbb{R}$. By the *open ball* with the center g and radius r we mean the set

$$B(g,r) = \{v \in V,\ \|v - g\|_V < r\}.$$

Definition A.12. Let V be a normed space and $S \subset V$. The set S is *open* in V if for every $g \in S$ there exists $r > 0$ such that $B(g,r) \subset S$. The set S is *closed* if its complement $V \setminus S$ is open.

A.2.2 L^p-spaces

These spaces have are very important for the solution of PDEs. These spaces were defined by Henri Léon Lebesgue who generalized the concept of Riemann integral. The entire concept of Lebesgue integral is based on the Lebesgue measure: we shall say that a set Ω_0 has zero Lebesgue measure in $\mathbb{R}^d$ if its d-dimensional measure is zero. For instance, the d-dimensional measure of a set Ω_0 consisting of a finite number of points or even of a countable infinite set of points, such as the set of rational numbers, is zero if $d \geq 1$.

Definition A.13. Let $f : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^d$ is an open measurable set. The Lebesgue integral of f over Ω is invariant with respect to the values of the function in zero-measure subsets of Ω. So

$$\int_\Omega f(\boldsymbol{x})\, d\boldsymbol{x} = \int_{\Omega \setminus \Omega_0} f(\boldsymbol{x})\, d\boldsymbol{x},\ \forall \Omega_0 \subset \Omega |\, |\Omega_0| = 0.$$

Because of this definition, the Riemann integral and the Lebesgue integral differ.

Definition A.14. Let $\Omega \subset \mathbb{R}^d$ be an open set. Consider the linear space V of measurable functions defined in Ω. For every $1 \leq p < \infty$ we define the *L^p-norm* in V as

$$\|f\|_p = \left(\int_\Omega |f(\boldsymbol{x})|^p\, d\boldsymbol{x} \right)^{\frac{1}{p}}.$$

The L^∞-norm in V is defined as

$$\|f\|_\infty = \operatorname{ess}\left(\sup_{\boldsymbol{x} \in \Omega} |f(\boldsymbol{x})| \right),$$

where the essential supremum is defined as

$$\operatorname{ess}\left(\sup_{\boldsymbol{x} \in \Omega} g(\boldsymbol{x}) \right) = \inf_{Z \subset \Omega, |Z| = 0} \left(\sup_{\Omega \setminus Z} g(\boldsymbol{x}) \right).$$

The spaces $L^p(\Omega)$ are defined as

$$L^p(\Omega) = \left\{ f \in V : \|f\|_p < \infty \right\}, \ \forall p : 1 \leq p < \infty$$

and

$$L^\infty(\Omega) = \left\{ f \in V : \operatorname{ess} \left(\sup_{\boldsymbol{x} \in \Omega} |f(\boldsymbol{x})| \right) < \infty \right\}.$$

A.3 Inner product spaces

An *inner product space* is a linear space equipped with an *inner product* $\langle \cdot, \cdot \rangle_V : V \times V \to \mathbb{C}$, where V is an inner product space. This structure associates each pair of vectors of the space with a scalar quantity known as inner product. An inner product space is also called a *pre-Hilbert space*. More formally:

Definition A.15. Let V be a real or complex linear space. An *inner product* in V is any function $\langle \cdot, \cdot \rangle_V : V \times V \to \mathbb{C}$ with the following properties:

1. for any $u \in V$, $\langle u, u \rangle_V \geq 0$ and moreover $\langle u, u \rangle_V = 0$ if and only if $u = 0$;
2. for any $u, v \in V$, $\langle u, v \rangle_V = \overline{\langle v, u \rangle}_V$;
3. for any $u, v, w \in V$ and all $a, b \in \mathbb{C}$ we have

$$\langle au + bv, w \rangle_V = a \langle u, w \rangle_V + b \langle v, w \rangle_V .$$

An *inner product space* V is a linear space over $\mathbb{C}$ with an inner product defined on it.

A.3.1 Hilbert spaces

For the definition of the Hilbert space it is necessary to define the Cauchy sequence first.

Definition A.16. Let V be a normed space. A sequence $\{v_n\}_{n=1}^{\infty} \subset V$ is a *Cauchy sequence* if for every $\mathbb{R} \ni \epsilon > 0$ there exists a natural number l such that for all natural numbers $m, n > l$

$$\|v_m - v_n\|_V < \epsilon.$$

Definition A.17. A normed space V (and so an inner product) is *complete* if every Cauchy sequence in V is a convergent sequence.

Definition A.18. Every complete inner product space is said to be a *Hilbert space*.

A.3.2 Bilinear forms and energetic spaces

The information that matters the most can be summarized in the following. We can associate every bilinear form on a Banach space V $a : V \times V \to \mathbb{R}$ with a unique linear operator $A : V \to V'$ defined by

$$(Au)(v) = \langle Au, v \rangle = a(u, v), \ \forall u, v \in V. \tag{A.1}$$

It can be shown that for each bilinear form $a : V \times V \to \mathbb{R}$ there is one and only one associated linear operator $A : V \to V'$. Some properties of bilinear forms are:

Definition A.19. Let V be a Hilbert space on $\mathbb{R}$, $a : V \times V \to \mathbb{R}$ a bilinear form and $A : V \to V'$ a linear operator related to $a(\cdot,\cdot)$ via (A.1). We can state the following:

1. $a(\cdot,\cdot)$ is *bounded* o if there exists a constant $C_a > 0$ such that $|a(u,v)| \leq C_a \|u\| \|v\|$, $\forall u, v \in V$ (*continuity*);
2. $a(\cdot,\cdot)$ is *V-elliptic* if there exists a constant $\tilde{C}_a > 0$ such that $a(v,v) \geq \tilde{C}_a \|v\|_V^2$, $\forall v \in V$;
3. $a(\cdot,\cdot)$ is *symmetric* if $a(u,v) = a(v,u)$, $\forall u, v \in V$.

The bilinear form can be used to define an inner product, and therefore a new space called Energetic space.

Definition A.20. Let V be a Hilbert space and $a : V \times V \to \mathbb{R}$ a bounded symmetric V-elliptic bilinear form. The bilinear form defines an inner product $\langle u, v \rangle_e = a(u,v)$ satisfying Definition A.15 called *energetic inner product*. The norm induced by the energetic inner product is

$$\|u\|_e = \sqrt{\langle u, u \rangle_e},$$

and it's called *energy norm*.

Once the energetic inner product is defined, we can define the energetic space. The reason for its name comes from physics, as in many systems the energy can be expressed as an energetic inner product.

Definition A.21. A subspace of an Hilbert space equipped with an energetic inner product is an *energetic space*.

A.3.3 Projections

Definition A.22. Let V be a linear space. An operator $P : V \to V$ is said to be a projection if it is both linear and idempotent ($P^2 = P$).

Lemma A.23. *Let V be a linear space. If V is a direct sum of two subspaces V_1 and V_2, $V = V_1 \oplus V_2$, then there exists a unique projection P so that $P(V) = V_1$, $(I - P)(V) = V_2$. Conversely, every projection operator P determines a decomposition of the space V into*

$$V = P(V) \oplus (I - P)(V).$$

An example of projection operator is the Lagrange interpolation

$$Pv = \sum_{i=0}^{n} v(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \ \forall v \in V.$$

Definition A.24. Let V be a Hilbert space. An operator $P : V \to V$ is said to be an orthogonal projection if it is linear, idempotent and if

$$(v - Pv, w)_V = 0, \ \forall v \in V, w \in P(V).$$

The space $P(V)$ is said to be the projection subspace.

Lemma A.25 reports an important fact about orthogonal projections: Pv is the closest element to $v \in V$ among all the elements in the projection space $P(V)$.

Lemma A.25. *Let V be a Hilbert space and W a closed subspace of V equipped with an orthonormal basis $B_W = \{w_1, \dots, w_n\}$. Let P be an orthogonal projection operator such that $P(V) = W$. Then for any $v \in V$ we have*

$$\|v - Pv\| = \inf_{w \in W} \|v - w\|.$$

A.4 Sobolev spaces

Definition A.26. Let $\Omega \subset \mathbb{R}^d$ be an open set, $k \geq 1$ an integer number and $p \in [1, \infty)$. we define

$$W^{k,p}(\Omega) = \{f \in L^p(\Omega) : D_w^\alpha f \text{ esists and lies in } L^p(\Omega) \text{ for all multi-indices } \alpha, \ |\alpha| \leq k\}.$$

For every $1 \leq p < \infty$ the norm $\|\cdot\|_{k,p}$ is defined as

$$\|f\|_{k,p} = \left(\int_\Omega \sum_{|\alpha| \leq k} |D_w^\alpha f|^p \, d\boldsymbol{x} \right)^{\frac{1}{p}} = \left(\sum_{|\alpha| \leq k} \|D_w^\alpha f\|_p^p \right)^{\frac{1}{p}}.$$

For $p = \infty$ we define

$$\|f\|_{k,\infty} = \max_{|\alpha| \leq k} \|D_w^\alpha f\|_\infty.$$

In the case $p = 2$ we abbreviate $W^{k,p}(\Omega) = H^k(\Omega)$.

It is important to note that:

Definition A.27. Let $\Omega \subset \mathbb{R}^d$ be an open set, $k \geq 1$ an integer number. The Sobolev space $W^{k,2} = H^k$, endowed with the inner product

$$\langle f, g \rangle_{k,2} = \int_\Omega \sum_{|\alpha| \leq k} D^\alpha f D^\alpha g d\boldsymbol{x} = \sum_{|\alpha| \leq k} \langle D^\alpha d, D^\alpha g \rangle_{L^2(\Omega)}$$

is a Hilbert space.

Switch definition.

Definition A.28. Let $\Omega \subset \mathbb{R}^d$ be an open set, $k > 1$ an integer number. The space $H_0^k(\Omega)$ is a subspace of the Hilbert space H^k and its definition is

$$H_0^k(\Omega) = \left\{ v \in H^k(\Omega) : D^\alpha v = 0 \text{ on } \partial\Omega \ \forall |\alpha| < k \right\}.$$

A.4.1 Distributions

Definition A.29. Let $\Omega \subset \mathbb{R}^d$ be an open set. The *space of distributions* is the space containing all the infinitely smooth functions with compact support, and can be defined by the writing

$$C_0^\infty(\Omega) = \{\varphi \in C^\infty(\Omega) : \operatorname{supp}(\varphi) \subset \Omega,\ \operatorname{supp}(\varphi) \text{ is compact}\},$$

where the *support* of the function φ is defined as

$$\operatorname{supp}(\varphi) = \overline{\{\boldsymbol{x} \in \Omega : \varphi(\boldsymbol{x}) \neq 0\}}$$

and it is always closed and bounded.

Example A.30. Consider a bounded domain $\Omega = (-1, 1) \subset \mathbb{R}$ and consider the functions

$$\varphi(x) = \cos(\pi x) + 1,\ \chi(x) = e^{-\frac{1}{1-x^2}}.$$

These functions are not distributions in Ω as

$$\operatorname{supp}(\varphi) = \operatorname{supp}(\chi) = [-1, 1] \not\subseteq \Omega.$$

However, the function χ can be extended by zero to be a distribution in the interval $(-1-\epsilon, 1+\epsilon)$, where $\epsilon > 0$. The same cannot be done for φ as it wouldn't be C^∞ anymore, and this is a requirement for being a distribution.

Remark A.31. As stated in Definition A.29, the support of a function $\varphi \in C_0^\infty(\Omega)$ is surely a closed set (as it is the closure of a set). This means any support can never touch the boundary of an open set Ω (like it was in Example A.30). This means for every $\varphi \in C_0^\infty(\Omega)$ there is a thin interval along the boundary $\partial\Omega$ where φ vanishes, as done in Example A.30.

A.4.2 Generalized integration by parts

Remark A.32. The formula of generalized integration by parts is often used when working with partial differential equations and in the weak formulation. Assume a bounded domain $\Omega \subset \mathbb{R}^d$ with Lipschitz-continuous boundary and assume

$$\boldsymbol{\nu}(\boldsymbol{x}) = (\nu_1, \nu_2, \ldots, \nu_d)^T(\boldsymbol{x})$$

is the outer normal to $\partial\Omega$. The Green's theorem for $H^1(\Omega)$-functions states:

Theorem A.33. *For every $u, v \in H^1(\Omega)$ it holds*

$$\int_\Omega \frac{\partial u}{\partial x_i} v d\boldsymbol{x} = -\int_\Omega u \frac{\partial v}{\partial x_i} d\boldsymbol{x} + \int_{\partial\Omega} uv\nu_i d\boldsymbol{S}.$$

Using Theorem A.33 it is possible to write, furthermore, the following lemma:

Lemma A.34. *For all $u \in H^1(\Omega)$ and $v \in H^2(\Omega)$ it is*

$$\int_\Omega u\Delta v d\boldsymbol{x} = -\int_\Omega \nabla u \nabla v d\boldsymbol{x} + \int_{\partial\Omega} u \frac{\partial v}{\partial \boldsymbol{\nu}} d\boldsymbol{S},$$

where

$$\frac{\partial v}{\partial \boldsymbol{\nu}} = \nabla v\left(\boldsymbol{x}\right) \cdot \boldsymbol{\nu}\left(\boldsymbol{x}\right),\ \boldsymbol{x} \in \partial\Omega.$$

For all $u \in \left[H^1\left(\Omega\right)\right]^d$ *and* $v \in H^1\left(\Omega\right)$ *it holds*

$$\int_\Omega \left(div\boldsymbol{u}\right) v d\boldsymbol{x} = -\int_\Omega \boldsymbol{u} \cdot \nabla v d\boldsymbol{x} + \int_{\partial\Omega} \left(\boldsymbol{u} \cdot \boldsymbol{\nu}\right) v d\boldsymbol{S}.$$

Appendix B

Notes of calculus

B.1 Continuity

In mathematics, a continuous function is a function for which, intuitively, small changes in the input result in small changes in the output. Otherwise, a function is said to be discontinuous. A continuous function with a continuous inverse function is called bicontinuous. An intuitive though imprecise (and inexact) idea of continuity is given by the common statement that a continuous function is a function whose graph can be drawn without lifting the chalk from the blackboard.

Definition B.1. A function f of variable $\boldsymbol{x}$ is *continuous* in $\boldsymbol{x}_0$ if

1. $f(\boldsymbol{x}_0)$ exists;
2. $\lim_{\boldsymbol{x}\to\boldsymbol{x}_0} f(\boldsymbol{x})$ exists;
3. $\lim_{\boldsymbol{x}\to\boldsymbol{x}_0} f(\boldsymbol{x}) = f(\boldsymbol{x}_0)$.

Definition B.2. A function $f : I \to \mathbb{R}$ of variable $\boldsymbol{x}$ is said to be k *times continuously differentiable in the interval* I, or *of class* $C^k(I)$, if its derivatives of order j, where $0 \leq j \leq k$, exists and are continuous functions for all $\boldsymbol{x} \in I$. A $C^\infty(I)$ function is a function that possesses continuous derivatives of any order.

B.2 Chain rule

The chain rule is the rule for the differentiation of the composite of two functions.

Definition B.3. Suppose $\boldsymbol{f} : \boldsymbol{x} \in \mathbb{R}^n \to \mathbb{R}^m$ is a function from the Euclidean n-space to the Euclidean m-space. Such a function is given by m real-valued component functions

$$f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n).$$

The partial derivatives of all these functions (if they exist) can be organized in an $m \times n$ matrix, the Jacobian matrix J_f as follows:

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

Possible notations are $J_f(x_1, x_2, \ldots, x_n)$, $\frac{\partial(y_1, y_2, \ldots, y_m)}{\partial(x_1, x_2, \ldots, x_n)}$ and $\frac{D\boldsymbol{f}}{D\boldsymbol{x}}$.

Theorem B.4. *Let ψ be a real-value function on (a, b) which is differentiable at $c \in (a, b)$, and let ϑ be a real-valued function defined on an interval l containing the range of ψ and $\psi(c)$ as an interior point. If ϑ is differentiable at $\psi(c)$, then*

$$(\vartheta \circ \psi)(x)$$

is differentiable at $x = c$, and

$$(\vartheta \circ \psi)'(c) = \vartheta'(\psi(c))\,\psi'(c).$$

It is possible to generalize this theorem to more variables and to partial derivatives using the definition of Jacobian matrix.

Theorem B.5. *Let $U \subset \mathbb{R}^m$ and $V \subset \mathbb{R}^n$ be open domains and let*

$$\boldsymbol{\psi} : V \to \mathbb{R}^l,\ \boldsymbol{\vartheta} : U \to V.$$

The chain rule takes the form

$$J_{\boldsymbol{\psi} \circ \boldsymbol{\vartheta}}(\boldsymbol{x}) = (J_{\boldsymbol{\psi}} \circ \boldsymbol{\vartheta})\, J_{\boldsymbol{\vartheta}}.$$

B.3 Integration by substitution

Theorem B.6. *Let U, V be open sets in $\mathbb{R}^n$ and $\varphi : U \to V$ an injective differentiable function with continuous partial derivatives, the Jacobian of which is nonzero for every $x \in U$. Then, for any real-valued, compactly supported, continuous function f, with support connected in $\varphi(U)$,*

$$\int_{\varphi(U)} f(\boldsymbol{v})\, d\boldsymbol{v} = \int_U f(\varphi(\boldsymbol{u}))\, |J_\varphi(\boldsymbol{u})|\, d\boldsymbol{u}.$$

B.4 Taylor expansion

The Taylor expansion is useful when facing the problem of approximating a function with polynomials in a point inside the domain. Let $f : (a, b) \to \mathbb{R}$ and $x_0 \in (a, b)$. If f is continuous in x_0 we can write (see Definition B.1 and using the definition of limit)

$$\lim_{x \to x_0} f(x) = f(x_0) \Leftrightarrow \lim_{x \to x_0} f(x) - f(x_0) = 0.$$

Using the Landau notation $o(\cdot)$ we can rewrite $f(x)$ as

$$f(x) = f(x_0) + o(1),\ x \to x_0$$

and, if f is differentiable in x_0 (by definition of differentiability)

$$\frac{f(x) - f(x_0)}{x - x_0} = f'(x),\ x \to x_0,$$

which can be rewritten again using the Landau notation

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + o(x - x_0),\ x \to x_0.$$

This way we were able to write $f(x)$ using a polynomial of degree 0 and a polynomial of degree 1, which can be thought of as approximations of $f(x)$. The same process can be continued again using polynomials of higher order. All of this can be summarized with the Taylor's theorem.

Theorem B.7. *If $n \geq 0$ is an integer and f is a function which is n times continuously differentiable on the closed interval $[a, x]$ and $n+1$ times differentiable on the open interval (a, x), then*

$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + ... + \frac{f^{(n)}(a)}{n!}(x - a)^n + R_n(x).$$

$R_n(x)$ is the remainder term, and several expressions are available to express it. The Taylor's theorem can be generalized to several variables as follows.

Theorem B.8. *Let B be a ball in $\mathbb{R}^n$ centered at a point a, and f be a real-valued function defined on the closure $\overline{B}$ having $n+1$ continuous partial derivatives at every point. Taylor's theorem asserts that for any $x \in B$*

$$f(x) = \sum_{|\alpha|=0}^{n} \frac{1}{\alpha!} \frac{\partial^\alpha f(a)}{\partial x^\alpha} (x - a)^\alpha + \sum_{|\alpha|=n+1} R_\alpha(x)(x - a)^\alpha$$

where the summation extends over the multi-indices α.

Index

Bibliography

[1] M. Aigner, C. Heinrich, B. Jüttler, E. Pilgerstorfer, B. Simeon, and A.-V. Vuong. Swept Volume Parameterization for Isogeometric Analysis. *Proceedings of the 13th IMA International Conference on Mathematics of Surfaces XIII*, 2009. 4.4

[2] Y. Bazilevs, V.M. Calo, J.A. Cottrell, J.A. Evans, T.J.R. Hughes, S. Lipton, M.A. Scott, and T.W. Sederberg. Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering*, In Press, Corrected Proof:–, 2009.

[3] Y. Bazilevs, L. Beirão da Veiga, J.A. Cottrell, T.J.R. Hughes, and G. Sangalli. Isogeometric analysis: Approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences (M3AS)*, 16(7):1031–1090, July 2006. 1, 4

[4] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):3 – 47, 1996.

[5] E. Bendìto, Á. Carmona, A.M. Encinas, and J.M. Gesto. Estimation of Fekete Points. *Journal of Computational Physics*, 225(2):2354–2376, 2007. 2.4.5.2

[6] M. Bertsch and R. Dal Passo. *Elementi di Analisi matematica*. Aracne, 2001.

[7] N. Bourbaki. *Elements of Mathematics*. Springer-Verlag.

[8] A. Campanaro and M.J. Collavo. Modellazione agli elementi finiti di disegni CAD multidimensionali utilizzando le NURBS, 2008.

[9] F. Cesari. *Comportamento non lineare delle strutture col metodo degli elementi finiti*. Pitagora, 1985.

[10] J.A. Cottrell, A. Reali, Y. Bazilevs, and T.J.R. Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5257 – 5296, 2006. John H. Argyris Memorial Issue. Part II.

[11] T. Dokken, V. Skytt, J. Haenisch, and K. Bengtsson. Isogeometric representation and analysis - bridging the gap between CAD and analysis. *47th AIAA Aerospace Sciences Meeting inlcuding The New Horizons Forum and Aerospace Exposition*, 2009.

[12] M.R. Dörfel, B. Jüttler, and B. Simeon. Adaptive isogeometric analysis by local h-refinement with T-splines. *Computer Methods in Applied Mechanics and Engineering*, In Press, Corrected Proof:–, 2008.

[13] P.J. Frey and P.-L. George. *Mesh Generation*. Hermes Science Publishing, 2000.

[14] Weimin Han and Xueping Meng. Error analysis of the reproducing kernel particle method. *Computer Methods in Applied Mechanics and Engineering*, 190(46-47):6157 – 6181, 2001.

[15] K. Höllig. *Finite Element Methods with B-Splines*. Society for Industrial and Applied Mathemaics, Philadelphia, PA, USA, 2003. 1

[16] A. Huerta and J. Donea. *Finite Element Methods for Flow Problems*. Wiley, April 2003.

[17] T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Mineola, 2000. 2.3.3

[18] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135 – 4195, 2005. 1, 4

[19] T.J.R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for nurbs-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, In Press, Corrected Proof:–, 2008. 4.6, 4.6.1

[20] B.M. Irons. Engineering Application of Numerical Integration in Stiffness Method. *Journal of the American Institute of Aeronautics and Astronautics*, 14:2035–2037, 1966. 2.2.1

[21] Y.W. Kwon and H. Bang. *The Finite Element Method Using Matlab*. CRC Press, 1997.

[22] F. Marcuzzi and M. Morandi Cecchi. The Best-Apprximation Weighted-Residuals Method for Finite Element Approximations. *MASCOT08-IMACS/ISGG*.

[23] A. Panizzo, L. Agostini, and L. Carlon. Implementazione in C (C++) di un codice agli elementi finiti per problemi ellittici bidimensionali. Project for the course "Calcolo Numerico", Jenuary 2008.

[24] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, 2nd edition edition, 1997. 3.5.6, 4.5.2

[25] A.D. Polyanin. *Handbook of linear partial differential equations for engineers and scientists*. CRC Press, 2002.

[26] W. Rudin. *Functional Analysis*. McGraw-Hill Science, 1991.

[27] S. Salsa. *Partial Differential Equations in Action: From Modelling to Theory*. Springer, 2008.

[28] T.W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. *ACM Trans. Graph.*, 22(3):477–484, 2003. 3.6

[29] Amit Shaw and D. Roy. NURBS-based parametric mesh-free methods. *Computer Methods in Applied Mechanics and Engineering*, 197(17-18):1541 – 1567, 2008.

[30] P. Šolín. *Partial Differential Equations and the Final Element Method.* John Wiley & Sons, Inc., 2006.

[31] P. Šolín, I. Dolezel, and K. Segeth. *Higher-Order Finite Element Methods.* CRC Pr I Llc, 2003. 2.2.1, 2.10.2

[32] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, volume 2nd. Springer-Verlag, 1992.

[33] I.C. Taig. Structural Analysis by the Matrix Displacement Method. *English Electric Aviation Report*, (S017), 1961. 2.2.1

[34] R.L. Taylor and O.C. Zienkiewicz. *The Finite Element Method*, volume 1. Butterworth-Heinemann, fifth edition, 2000. 2.8, 2.10.2

[35] The MathWorks. *Optimization toolbox User Guide*, 2008.

[36] M.E. Tylor. *Partial Differential Equations I.* Springer.

[37] W. Gander and W. Gautschi. Adaptive Quadrature - Revisited. *BIT*, 40(1):84–101, March 2000. (document), 4.12, 4.7.5, 4.14

[38] Y. Zhang, Y. Bazilevs, S. Goswami, C.L. Bajaj, and T.J.R. Hughes. Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. *Computer Methods in Applied Mechanics and Engineering*, 196(29-30):2943 – 2959, 2007.

[39] O.C. Zienkiewicz, J.Z. Zhu, and N.G. Gong. Effective and practical h-p-version adaptive analysis procedures for the finite element method. *International Journal for Numerical Methods in Engineering*, (28):879–891, 1989. 2.10.2