NTNU – Trondheim
Norwegian University of
Science and Technology

On Local Refinement in Isogeometric Analysis

A comparative study on Classical
Hierarchical, Truncated Hierarchical and LR
B-splines

Filippo Remonato

Master of Science in Mathematics (for international students)
Submission date: June 2014
Supervisor: Trond Kvamsdal, MATH
Co-supervisor: Kjetil Andre Johannessen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This Master Thesis is the final assignment for the International Master Programme (MSc) in Mathematics. While an effort has been made to present the topics in a self-contained way, the reader is expected to have some prior knowledge in the Finite Element Method.

An extract of this work has been submitted for publication to the journal *Computer Methods in Applied Mechanics and Engineering*, for the *Isogeometric Special Issue*. The results contained in this work have also been presented at the IGA2014 conference, held at the University of Texas at Austin on January 8-10, 2014.

I would like to acknowledge here the support, supervision, and guidance of Trond Kvamsdal, professor at the Norwegian University of Science and Technology - NTNU, as well as the collaboration I had with Kjetil André Johannessen, PhD student at NTNU, which helped me a great deal with several hours of discussions and suggestions, from the reference papers to read to the code development in Matlab and the interpretation of the results we obtained. Lastly, I would like to thank my parents for having always supported me in my choices, and especially in this wonderful Norwegian adventure.

Abstract

Smooth spline functions such as B-splines and NURBS are already an established technology in the field of computer-aided design (CAD) and have in recent years been given a lot of attention from the computer-aided engineering (CAE) community. The advantages of local refinement are obvious for anyone working in either field, and as such, several approaches have been proposed. Three strategies are the Classical Hierarchical B-splines, the Truncated Hierarchical B-splines and the Locally Refined B-splines. We will in this paper present these three frameworks and highlight similarities and differences between them. In particular, we will look at the function space they span and the support of the basis functions. We will then analyse the corresponding stiffness and mass matrices in terms of sparsity patterns and conditioning numbers. We show that the basis in general do not span the same space, and that conditioning numbers are comparable. Moreover we show that the weighting needed by the Classical Hierarchical basis to maintain partition of unity has extreme implications on the conditioning numbers.

Contents

0

Introduction

Isogeometric Analysis (IGA) is a relatively recent technique for the discretization and solution of Partial Differential Equations (PDE), presented for the first time by Hughes, Cottrell and Bazilev in [14].

At the present day, most of industry-made objects are designed through the use of Computer Aided Design (CAD). These techniques use particular geometrical functions to graphically represent objects; the most used being the NURBS functions, an extension of the B-splines. One of the main features of the Isogeometric methods is to provide an *exact* representation of the geometry of the object of the analysis; this is done through the so-called *isoparametric* approach, which consists in using the same CAD functions used to generate the object for the analysis of the object itself. In this way, the exact representation of the object is guaranteed since the very first mesh, independently of the characteristic size h of the mesh's elements, and successively maintained at each stage of mesh refinement.

B-splines and NURBS function spaces contain, as a special case, the space of piecewise polynomial functions used by the classical Finite Element Methods (FEM). Due to this, Isogeometric Analysis can be seen as a generalization of the FEM methods by the use of functions with high regularity. This increased smoothness in the basis functions yields important advantages, as shown in [4].

With both FEM and IGA methods a partial differential equation is solved via a Galerkin's approach, which consists in the construction of suitable function spaces within which to search for the solution. The main difference between those methods lies in the different basis functions chosen to both calculate the numerical solution as well as constructing the mesh on the physical domain. FEM methods use piecewise polynomial functions for the function space, and polygonal mesh elements, like triangles (2D) or tetrahedra (3D). IGA methods use, as we said, CAD functions like NURBS or B-splines for both the function space and the mesh. This allows for the construction of elements with curved edges and consequently an exact representation of the geometry of the object, which was created using the same functions.

Since their introduction, IGA methods have evolved in different directions and several different basis functions have been proposed, as well as refinement schemes. In particular, in the last few years there has been an increasing attention of the scientific community for local refinement techniques. The wide number of publications and conferences on IGA shows how these new methods are raising a strong interest in both the design and egineering/numerical analysis community, in particular for the possibility of a tighter interaction between the two fields.

0.1 Isogeometric Analysis: motivations and aim

The Finite Element analysis for engineering started in the 1950-1960 and was used mainly in the aerospace industry, which in those years was experiencing a fast expansion. At the end of the '60s the first commercial softwares for analysis were produced and, as a consequence, the finite element methods spread also in other scientific areas. As of today, they are considered the standard in most mathematical analysis required by the industry.

Despite the fact that geometry is the structure underlying analysis, CAD graphics were developed later, in the 1970. This is probably the main reason why the geometrical representations of the objects are so different between the FEM and CAD communities. It is a generally accepted fact that the invention of modern computer graphics comes from the works of two French automotive engineers, Pierre Bezier (Renault) and Paul de Faget de Casteljau (Citroën). Bezier in 1966 presented his technique for the construction of curves and surfaces based on the use of Bernstein's polynomials. De Casteljau already developed similar techniques in 1959, but his works were not published. The term 'spline' instead was used the first time in 1946, in a work by Schoenberg, which explained how they could be used to approximate shapes and functions. During the '70s a large number of publications gave a strong impulse to the CAD geometry, in particular the PhD thesis of Reisenfeld regarding B-splines (1972) and the one of Verspille which presented NURBS (1975).

Today most of the objects produced by the industry are designed through the use of softwares based on CAD geometry: think for example of the famous AutoCAD, or Maya; or more specific softwares like Rhino or Catia, developed by Boeing and considered the state-of-the-art for engineering. Design blueprints have long since stopped being drawings on paper, and became large CAD files. Executing a mathematical simulation on these projects requires the conversion from the CAD geometry description to one suitable for analysis purposes, the mesh creation and then the computation using a finite element method. This process is all but straightforward, and engineering projects are becoming each day more and more complex. On the other hand, the design process itself depends on a variety of mathematical analysis results and simulations; it is therefore inevitable and essential a continuous flow of data between the CAD world and the FEA one. The main problem is, indeed, that analysis-suitable models are not made directly by the CAD geometry with which the object is designed, and this causes major consequences on the communication process between the two technologies. This process has been analysed by Ted Blacker, of Sandia National Laboratories; the results are presented in Figure 1.

As we can see, just the creation of an analysis-suitable geometry uses 60% of the total time of the process; 20% is used by mesh construction and manipulation, and only the last 20% is used on actual analysis. This situation seems to be very common in today's industry and there is a strong desire for a change. For example, in the automotive industry a mesh for an entire vehicle takes around four months of computation time, while during the development stages the design is updated almost every day. This impractical difference in the times is one of the main causes of the small use of mathematical analysis as a tool to optimize the design process.

The increasing complexity of the industrial projects and engineering techniques are now more than ever pressing for faster analysis results, which can be used to optimize each stage of the design process. In addition, we note that the mesh generated by the classical finite elements procedure is only an approximation of the original CAD geometry, which we view as the exact model to represent. This difference, however small, can cause

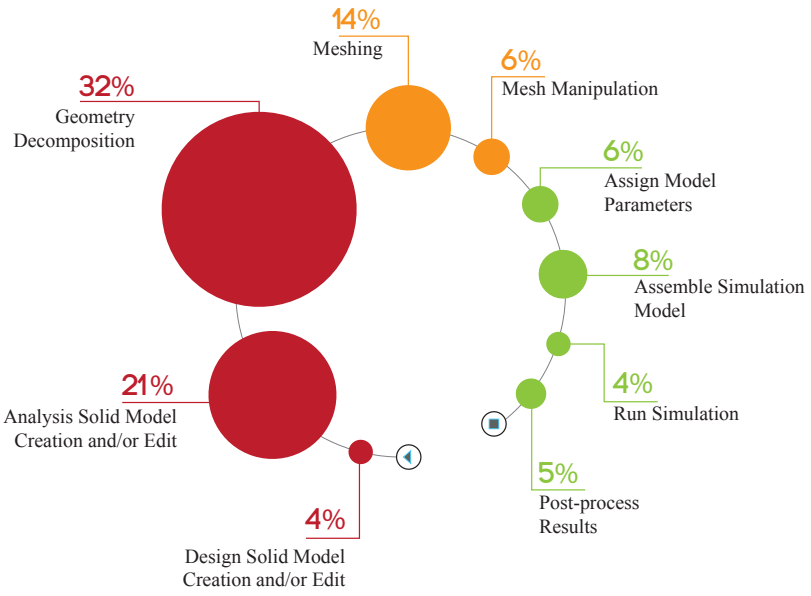


Figure 1: Results of the study on the time used by each phase of the analysis process at Sandia National Laboratories.

severe problems of accuracy in the numerical solution, especially for particularly unstable problems.

To overcome the above problems a shift is therefore necessary, from the classical finite elements methods to an analysis based on the same geometry and functions used to design the objects. This is the foundation point of Isogeometric Analysis. The main aims of this technique are to be able to retrieve the meshes directly from the CAD representations; to have an exact description of the geometry regardless of how coarse the mesh is, and then to simplify the refinement processes in both the mesh and the functions basis used for analysis.

A fundamental constraint of traditional NURBS is that they are tensor product and lack the potential for local refinement. The need for local refinement have always been an issue, and several proposed solutions to local refinement have been derived such as T-splines [27],[26], Hierarchical B-splines [9],[18], Truncated Hierarchical B-splines [11] and Locally Refined B-splines [6]. The use of these techniques in CAD allows for more freedom since it is often enough that a forward mapping exist and can be efficiently manipulated or evaluated. With their applications into isogeometric analysis [1], [7], [23], [29], [28], [2], [24], and [15] came new requirements of the basis. Linear independence [25], stability [12] and partition of unity [11] became center topics of research as isogeometric analysis is impractical or sometimes impossible without these properties. The research is ongoing for most of these basis and the community has yet to settle on a single technology which encompass every desired property without restrictions on the mesh.

With many different technologies addressing the same problem of local refinement, it is to be expected that there is overlap. We hope to shed some light on this topic by presenting some of these spline families, highlighting similarities and differences between them. In particular we will be looking at the Classical and Truncated Hierarchical, and the Locally Refined B-splines.

0.2 Aim and outline of the paper

We have organized this paper as follows:

Chapter 1, which is an excerpt from [22], provides an introduction to the Finite Element Theory. After a very brief presentation of Sobolev and Hilbert spaces, to settle the notation used in he rest of the paper, we present three classical problems where the Galerkin's approach is used: The Poisson problem, the Least Squares approximation problem, and Helmholtz equation. The Poisson problem is the standard model problem for elliptic partial differential equations. The procedure to derive its weak formulation and the corresponding discrete formulation and solution is described in detail. The Least Squares problem and the Helmholtz equation are presented more briefly, since several of the steps have already been explained in the Poisson case.

Chapter 2, also an excerpt from [22], presents an introduction to the geometry of CAD functions. First Bezier curves are introduced, which are the foundation upon which more complex objects, like B-splines and NURBS, are constructed. We then present the general theory of B-splines and also provide a short overview of the three standard refinement methods that B-splines come naturally equipped with: h-refinement, p-refinement and k-refinement.

Chapter 3 presents three state-of-the-art local refinement frameworks that have the potential to become the standard in the field: Classical Hierarchical, Truncated Hierarchical and LR B-splines. The refinement strategies presented in the previous chapter have the disadvantage of being *global*, in the sense that the refinement will affect large portions of the mesh (in case of h-refinement) or all basis functions (in case of p-refinement). The strategies presented here, instead, allow for a *local* refinement of the mesh. We start by fixing the notation for the rest of the paper and then proceed to present the Classical Hierarchical B-splines. The general idea is that the basis of functions defined on a mesh can be constituted of several different "levels", which provide different resolutions where needed. Unfortunately, this technique results in several functions having support over the same element, and consequently a denser stiffness or mass matrix. To solve this problem, the Truncated Hierarchical B-splines were introduced. We then finish this chapter by presenting the LR B-splines technology, the last of the local refinement schemes we considered.

Chapter 4 present the results we obtained, both from a *qualitative* (more theoretical) and from a *quantitative* (more numerical) perspective. We will show that there are several differences between the three approaches presented in Chapter 3, and provide numerical examples.

Chapter 5 presents some more interesting results we have found during our tests, but for which more research is required in order to fully understand their meaning and consequences.

We conclude this paper by giving our conclusions upon our results in Chapter 6.

1

Finite Element Theory

In this chapter, which is an excerpt from [22], is presented an introduction to the Finite Element theory. A thorough dissertation can be found in many good sources like [16, 21, 3, 13].

While the finite element method can be applied to a variety of problems, we will focus mainly on *Boundary Value* problems. The boundary value problems, or BVP, are one of the two main classes of problems that appear in the solution of differential equations, both ordinary and partial (the other class is the *Initial value* problems).

From a mechanical point of view we can imagine both categories as representing physical systems of which one wants to know the configuration or their evolution in time. BVP describe problems where some conditions on the boundary of the domain are known, and one wants to know the configuration assumed by the system: think for example of an elastic rope fixed at two ends. The position of the ends of the rope is known, and we want to know which shape the rope will assume due to the gravitational forces acting oh it. IVP instead prescribe not only some boundary conditions but also some initial conditions, and then the evolution of the system in time is the unknown. For example we can consider a metal bar placed in a certain position in space and we also know the initial temperature distribution inside the bar. If we then apply a heat source we are interested in studying the evolution of the temperature inside the bar as time passes.

From a mathematical point of view these problems are described as a relation between the the unknown function u and its derivatives which is of the form $\mathcal{F}(x, u, u_{x1}, u_{x2}, \ldots) = 0$. One then wants to know which function u satisfies this equation. Typically a solution, if it exists, is not unique. Indeed, most of the times if a solution exists, then there are infinite solutions. The application of the prescribed conditions, whether initial or on the boundary, is required to obtain a unique solution.

Several methods to solve these kind of problems have been developed; we will in this chapter present the *Galerkin Method*, and also give a brief glimpse on the solution as a minimization problem. Galerkin's method is considered the standard approach for finite elements methods and is based on rewriting a problem (D) in a more "relaxed" form, called *variational* or *weak* form, (V). We will prove that under appropriate conditions these two formulations are equivalent.

1.1 Sobolev and Hilbert spaces

The solution of boundary value problems, an in particular the variational formulation, make use of functions belonging to Sobolev and Hilbert spaces. We give here a short

introduction to these spaces; we will start recalling some concepts from Linear Algebra, and then define the Hilbert space L^2 and the Sobolev spaces $W^{m,p}$ of which the space H^1 is a special case. A detailed dissertation on normed and Hilbert spaces can be found in [19].

Let V be a linear space, we say that L is a *linear form* on V if $L : V \longrightarrow \mathbb{R}$ and is linear, i.e.

$$L(\alpha v + \beta u) = \alpha L(v) + \beta L(u) \qquad \forall\, \alpha,\, \beta \in \mathbb{R}$$

We say that $a(\cdot\,,\cdot)$ is a *bilinear form* on $V \times V$ if $a : V \times V \longrightarrow \mathbb{R}$ and is linear in both arguments. In particular, a bilinear form is said to be *symmetric* if

$$a(u, v) = a(v, u) \qquad \forall\, u, v \in V$$

A bilinear symmetric form is called an *inner product* on V if

$$a(v, v) > 0 \qquad \forall\, v \in V,\ v \neq 0$$

The *norm* $\|\cdot\|$ associated with an inner product is given by

$$\|v\| = (a(v, v))^{1/2} \qquad \forall\, v \in V$$

moreover, the *Cauchy-Schwartz Inequality* holds:

$$|a(u, v)| \leqslant \|u\|\,\|v\| \qquad \forall\, u, v \in V$$

We then say that a space V is *complete* respect to the norm $\|\cdot\|$ if all Cauchy sequences in V converge respect to that norm.

Definition 1. Let V be a normed linear space, i.e. a linear space with a norm induced by an inner product. V is called a *Hilbert Space* if it is complete.

Let now I be a bounded domain. The set

$$L^2(I) = \{v : I \to \overline{\mathbb{R}} \mid \textstyle\int_I v^2 \,\mathrm{d}x < \infty\} \tag{1.1}$$

is the space of square-integrable functions. L^2 is a Hilbert space with the inner product defined as

$$(u, v) = \int_I uv \,\mathrm{d}x$$

The corresponding norm is called L^2-norm and is given by

$$\|v\|_{L^2} = \left(\int_I v^2 \,\mathrm{d}x\right)^{1/2}$$

The above can be generalized for an arbitrary p by defining

$$L^p(I) = \{v : I \to \overline{\mathbb{R}} \mid \textstyle\int_I |v|^p \,\mathrm{d}x < \infty\}$$

with the associated norm

$$\|v\|_{L^p} = \left(\int_I |v|^p \,\mathrm{d}x\right)^{1/p}$$

We now want to define the Sobolev Spaces. We introduce a notation called *multi-index* which simplifies the expressions we will need. We will call *multi-index vector* a vector $\boldsymbol{\alpha} = (\alpha_1 \dots \alpha_n) \in \mathbb{N}^n$. Let v be a function defined on I, then we can write its derivatives as

$$D^{\boldsymbol{\alpha}}\, v(\boldsymbol{x}) = \frac{\partial^{|\boldsymbol{\alpha}|} v(\boldsymbol{x})}{\partial x_1^{\alpha_1}\, \partial x_2^{\alpha_2} \dots \partial x_n^{\alpha_n}}$$

Definition 2. Let $m \in \mathbb{N}$. The *Sobolev Space* $W^{m,p}$ is defined as

$$W^{m,p}(I) = \{v \in L^p(I) \mid D^{\boldsymbol{\alpha}} v \in L^p(I) \ \ \forall \boldsymbol{\alpha} : |\boldsymbol{\alpha}| \leqslant m\} \tag{1.2}$$

where $|\boldsymbol{\alpha}| = \sum_{k=1}^{n} \alpha_k$.

A Sobolev Space $W^{m,p}$ is then a subset of L^p made by all the functions whose derivatives are also in L^p. On $W^{m,p}$ it is possible to define the inner product as

$$(u, v)_{W^{m,p}} = \sum_{|\boldsymbol{\alpha}| \leqslant m} \int_I D^{\boldsymbol{\alpha}} u \, D^{\boldsymbol{\alpha}} v \, \mathrm{d}x \tag{1.3}$$

with the associated norm

$$\|v\|_{W^{m,p}} = \left(\sum_{|\boldsymbol{\alpha}| \leqslant m} \int_I |D^{\boldsymbol{\alpha}} v|^p \, \mathrm{d}x \right)^{1/p} \tag{1.4}$$

By choosing $p = 2$ and $m = 1$ we obtain the space $W^{1,2}$ which is generally indicated with H^1. We will see that this space is the natural setting to define many of the variational problems we will face in the next sections. From (1.2) we can write

$$H^1(I) = \{u \in L^2(I) \mid D^{\boldsymbol{\alpha}} u \in L^2(I) \ \ \forall \boldsymbol{\alpha} : |\boldsymbol{\alpha}| \leqslant 1\} \tag{1.5}$$

and from (1.3) and (1.4) we have the inner product and norm on H^1:

$$(u, v)_{H^1} = \int_I (uv + \nabla u \cdot \nabla v) \, \mathrm{d}x \tag{1.6}$$

$$\|u\|_{H^1} = \left(\int_I (u^2 + |\nabla u|^2) \, \mathrm{d}x \right)^{1/2} \tag{1.7}$$

We will often need to consider a subset of H^1, indicated with H_0^1 and defined as

$$H_0^1(I) = \{u \in H^1(I) \mid u = 0 \ \text{ su } \partial I\} = 0 \tag{1.8}$$

with the same inner product and norm as in H^1.

1.2 Poisson Problem

We will now illustrate the Finite Element Method applying it to the Poisson Problem, following the approach given in [4]. The exposition of the Poisson problem will be performed with an abundance of details, while Sections 1.3 and 1.4 will synthetically present the Least Squares problem and the Helmholtz equation, since many steps will be fundamentally the same.

Poisson equation is the classical model problem for Elliptic PDEs. It arises in several engineering problems like elastic membranes or magnetic fields and also appears as an important part of more complicated problems like Navier-Stokes. Given a domain $\Omega \subset \mathbb{R}^2$ and a continuous function $f : \Omega \to \mathbb{R}$, we want to find a function $u : \Omega \to \mathbb{R}$ such that

$$-\Delta u = f \quad \text{in } \Omega \tag{1.9}$$

and satisfies certain prescribed conditions on the boundary of the domain $\Gamma = \partial\Omega$.

With the common notation used in these situations, the general Poisson Problem can be expressed as

$$\begin{cases} \Delta u + f = 0 & \text{in } \Omega \\ u = g & \text{on } \Gamma_D \\ \nabla u \cdot \hat{\boldsymbol{n}} = h & \text{on } \Gamma_N \end{cases} \tag{D}$$

where $\hat{\boldsymbol{n}}$ is the outward normal vector, Ω is the domain in which the problem is defined and Γ_D and Γ_N form a partition of the boundary of Ω. The conditions $u = g$ and $\nabla u \cdot \hat{\boldsymbol{n}} = h$ are called *Dirichlet condition* and *Neumann condition*, respectively. The problem written as in (D) is called **strong formulation** of the Poisson problem.

A very useful formula in this setting is the following *Green's Formula*:

$$\int_\Omega \nabla v \cdot \nabla w \, \mathrm{d}x = \int_\Gamma v \, \nabla w \cdot \hat{\boldsymbol{n}} \, \mathrm{d}s - \int_\Omega v \, \Delta w \, \mathrm{d}x \tag{1.10}$$

And we define the *space of possible solutions*

$$\mathcal{S} = \{u \mid u \in H^1(\Omega),\ u_{|\Gamma_D} = g\} \tag{1.11}$$

along with the *space of test functions*:

$$\mathcal{V} = \{w \mid w \in H^1(\Omega),\ w_{|\Gamma_D} = 0\} \tag{1.12}$$

At this point we are ready to derive the variational formulation of the Poisson problem. We start by multiplying the equation of problem (D) with a generic test function $w \in \mathcal{V}$ and then integrate over the whole domain to obtain

$$\int_\Omega w \, \Delta u \, \mathrm{d}x + \int_\Omega w f \, \mathrm{d}x = 0$$

we can now use equation (1.10) on the first integral to obtain

$$\int_\Gamma w \, \nabla u \cdot \hat{\boldsymbol{n}} \, \mathrm{d}s - \int_\Omega \nabla w \cdot \nabla u \, \mathrm{d}x + \int_\Omega w f \, \mathrm{d}x = 0 \tag{1.13}$$

due to the linearity of the integral respect to the integration domain, we can split the second term in (1.13) in the two parts on Γ_D and Γ_N. Since $w = 0$ on Γ_D, that integral vanishes, and we derive the **weak formulation**:

Find $u \in \mathcal{S}$ such that $\forall\, w \in \mathcal{V}$:

$$\int_\Omega \nabla w \, \nabla u \, \mathrm{d}x = \int_\Omega w f \, \mathrm{d}x + \int_{\Gamma_N} w \, h \, \mathrm{d}s \tag{1.14}$$

Defining

$$a(w, u) = \int_\Omega \nabla w \cdot \nabla u \, \mathrm{d}x$$

$$L(w) = \int_\Omega w f \, \mathrm{d}x + \int_{\Gamma_N} w \, h \, \mathrm{d}s$$

we can write (1.14) in the compact form

$$a(w, u) = L(w) \qquad \forall\, w \in \mathcal{V} \tag{V}$$

We see that $a(\cdot\,,\cdot)$ is a symmetric positive definite bilinear form while $L(\cdot)$ is a linear functional. It's important to note that, while the problem in the strong form D required u to have defined second derivatives, the variational formulation V requires only the first derivatives of u to be square-integrable. Due to this relaxation in the requirements on u, the variational formulation is often called *weak* formulation.

Due to the way the variational formulation is constructed, it is clear that if u is a solution of D, then it is also a solution of V. Clearly, the converse is true only if the weak solution has enough regularity.

1.2.1 Galerkin approach

The Galerkin approach consist of defining proper finite-dimensional subspaces of $\mathcal{S}$ and $\mathcal{V}$. These subspaces, called $\mathcal{S}_h$ and $\mathcal{V}_h$ are linked to the isoparametric base we're using and they are such that

$$\mathcal{S}_h \subset \mathcal{S}$$
$$\mathcal{V}_h \subset \mathcal{V}$$
$$dim\,\mathcal{S}_h,\ dim\,\mathcal{V}_h < \infty$$

We can further characterize the space $\mathcal{S}_h$ observing that given $g_h \in \mathcal{S}_h$ such that $g_h|_{\Gamma_D} = g$, for each $u_h \in \mathcal{S}_h$ there exist a (unique) $v_h \in \mathcal{V}_h$ such that

$$u_h = v_h + g_h \tag{1.15}$$

Where g is called a *lifting function*. We note that observation is not true for any possible function g. For many different functions g what is stated above holds, but there are also a number of cases where we cannot achieve an "exact" equality and we will have to use an approximation of g. A thorough discussion of this, however, is out of the scope of this thesis. For our purposes it's sufficient to assume that the above statement holds.

Using the Galerkin approach the problem can now be formulated as:

Find $u_h = v_h + g_h$, with $v_h \in \mathcal{V}_h$, such that $\forall\, w_h \in \mathcal{V}_h$

$$a(w_h, u_h) = L(w_h) \tag{V_G}$$

Using the bilinearity of $a(\cdot\,,\cdot)$ we can then write

$$a(w_h, v_h) = L(w_h) - a(w_h, g_h) \tag{1.16}$$

We note that the right side of the equation involves only known quantities while the unknown, now v_h, is contained only in the left side.

As in classical finite elements, the finite-dimensional nature of the Galerkin approach allows us to write our problem as a system of algebraic equations.

Let $\{N_1 \dots N_{n_{np}}\}$ be the basis functions[1] defined on the parametric space $\hat{\Omega}$, where n_{np} is the total (global) number of basis functions. We then define $\mathcal{S}_h$ as

$$\mathcal{S}_h = span\{N_1 \dots N_{n_{np}}\}$$

i.e., the space of all possible functions defined as a linear combination of the basis functions.

[1] In our case these are the B-Spline basis functions, which are presented in Chapter 2. For now, it is sufficient to think of some generic functions constituting a basis.

As we have seen in the CAD section, many of these functions are zero on the boundary of the domain. We can assume a numbering $n_{eq} < n_{np}$ for these functions, such that

$$N_i|_{\Gamma_D} = 0 \qquad \forall\, i = 1 \dots n_{eq}$$

We can then expand all function in $\mathcal{V}_h$ in terms of the basis functions: for any $v_h \in \mathcal{V}_h$, we can write

$$v_h = \sum_{j=1}^{n_{eq}} u_j\, N_j \tag{1.17}$$

In principle we could do the same for g_h. With the observation given above we know that the first n_{eq} coefficients have no effect on the value of g_h on the boundary, we can then set $g_1 = g_2 = \ldots = g_{n_{eq}} = 0$ and write

$$g_h = \sum_{i=n_{eq}+1}^{n_{np}} g_i\, N_i \tag{1.18}$$

From (1.15) we have that any $u_h \in \mathcal{S}_h$ can then be written as

$$u_h = \sum_{j=1}^{n_{eq}} u_j\, N_j + g_h \tag{1.19}$$

Since in (1.15) the only unknown was v_h, we see that now the unknowns are the coefficients d_j. Lastly, we recall that (V_G) must be satisfied for all test functions $w_h \in \mathcal{V}_h$. We can achieve this by systematically selecting

$$w_h = N_i \text{ for } i = 1 \dots n_{eq}$$

Substituting the above expressions into (1.16) we have that

$$a\left(N_i\ ,\ \sum_{j=1}^{n_{eq}} u_j\, N_j\right) = L\left(N_i\right) - a\left(N_i\ ,\ g_h\right)$$

must hold $\forall\, i = 1 \dots n_{eq}$. Using the linearity of a and L we derive

$$\sum_{i=1}^{n_{eq}} a(N_i, N_j)\, u_j = L(N_i) - a(N_i, g_h) \qquad \forall\, i = 1 \dots n_{eq} \tag{1.20}$$

Defining

$$\begin{aligned} A_{i,j} &= a(N_i, N_j) \\ b_i &= L(N_i) - a(N_i, g_h) \end{aligned}$$

for $i, j = 1 \dots n_{eq}$ and the corresponding matrix and vectors

$$\begin{aligned} A &= [A_{i,j}] \\ \boldsymbol{b} &= [b_i] \\ \boldsymbol{u} &= [u_i] \end{aligned}$$

we see that equation (1.20) can be written as the system of algebraic equations

$$A\boldsymbol{u} = \boldsymbol{b} \tag{1.21}$$

Solving this system we obtain the coefficients vector $\boldsymbol{u}$ and we can then write the solution of (V_G) as

$$u_h = \sum_{j=1}^{N_{eq}} u_j \, N_j + \sum_{i=n_{eq}+1}^{n_{np}} g_i \, N_i \tag{1.22}$$

Due to historical reasons, the matrix A is called *Stiffness Matrix*, and the vector $\boldsymbol{b}$ is called *Load Vector*.

1.2.2 Minimization problem

We will now show how problem (V) is naturally associated with a minimization problem (M). Given a linear functional $F : H^1 \longrightarrow \mathbb{R}$ defined as

$$F(w) = \frac{1}{2}\, a(w, w) - L(w)$$

we want to find the function u that minimizes F, i.e. such that

$$F(u) \leqslant F(w) \qquad \forall\, w \in \mathcal{V} \tag{M}$$

We now prove the equivalence of the problem in the weak form and the minimization problem. Let u be a solution of (V), and let w be a generic test function. Letting $v = w - u$ we have that $v \in H^1$ and $w = u + v$, therefore the following holds:

$$\begin{aligned} F(w) = F(u+v) &= \frac{1}{2}\, a(u+v, u+v) - L(u+v) \\ &= \frac{1}{2}\, a(u,u) + \frac{1}{2}\, a(v,v) + a(u,v) - L(u) - L(v) \\ &= \frac{1}{2}\, a(u,u) - L(u) + \frac{1}{2}\, a(v,v) \\ &\geqslant F(u) \end{aligned}$$

Since this inequality holds for every test function w, we have that u is the minimizer of F and therefore a solution of (M).

Viceversa, let now u be a solution of (M); then $\forall \epsilon > 0$ we have that

$$F(u) \leqslant F(u + \epsilon\, w) \qquad \forall\, w \in \mathcal{V}$$

therefore the function

$$\begin{aligned} g(\epsilon) = F(u + \epsilon\, w) &= \frac{1}{2}\, a(u + \epsilon\, w, u + \epsilon\, w) - L(u + \epsilon\, w) \\ &= \frac{1}{2}\, a(u,u) + \frac{\epsilon^2}{2}\, a(w,w) + \epsilon\, a(u,w) - L(u) - \epsilon\, L(w) \end{aligned}$$

has a minimum for $\epsilon = 0$. In the minimum point the first derivative of g must vanish, therefore we have

$$g'(0) = a(u,w) - L(w) = 0$$

hence u is a solution of the variational problem (V). Such a solution is unique: if we let u_1 and u_2 be two distinct solutions of (V) then it must be

$$\begin{aligned} a(u_1, w) &= L(w) \\ a(u_2, w) &= L(w) \end{aligned} \qquad \forall\, w \in \mathcal{V} \tag{1.23}$$

Since both u_1 and u_2 must satisfy the same boundary conditions, we can choose $w = u_1 - u_2$. We can then subtract the two equation in (1.23) from each other. Since the measure of Γ is zero, we obtain

$$\int_{\overline{\Omega}} (\nabla u_1 - \nabla u_2)^2 = 0$$

$$\Rightarrow\ \nabla u_1 - \nabla u_2 = 0 \text{ in } \overline{\Omega}\ \Rightarrow\ \nabla(u_1 - u_2) = 0 \text{ in } \overline{\Omega}\ \Rightarrow\ u_1 - u_2 = \mathit{cost} \text{ in } \overline{\Omega}$$

where $\overline{\Omega}$ is the closure of Ω. Again, from the boundary conditions we know that $u_1 = g = u_2$ on Γ_D and so $u_1 - u_2 = 0$ on Γ_D. From this follows that $u_1 - u_2 = 0$ in $\overline{\Omega}$, and so the two solutions coincide.

To summarize the above, we have shown that a solution of the problem (D) is also a solution of the variational formulation (V), which is equivalent to the minimization problem (M). Symbolically we have

$$\text{(D)} \quad \Rightarrow \quad \text{(V)} \quad \Leftrightarrow \quad \text{(M)}$$

1.3 Least Squares fitting

Performing a least-square fit of a surface is a problem often encountered in a geometrical setting. In this case, given a smooth continuous function $f : \Omega \to \mathbb{R}$ we are searching for a function $u_h \in \mathcal{V}_h$ such that $\|u_h - f\|_{L^2}$ is as small as possible.

It is possible to show that the solution u_h is the L^2-projection of f and

$$u_h = \underset{u \in \mathcal{V}_h}{\operatorname{argmin}} \|u - f\|_{L^2} \iff \int_\Omega u_h\, v_h \,\mathrm{d}A = \int_\Omega f\, v_h \,\mathrm{d}A \quad \forall v_h \in \mathcal{V}_h$$

Note that in this case the value of f on $\partial\Omega$ is known, therefore the constraint are of Dirichlet-type.

Applying the same procedure as in the Poisson's equation case we are able to write the problem as searching for the solution of a linear system of equations

$$M\boldsymbol{u} = \boldsymbol{b}$$

where now the matrix M is called *Mass Matrix* and is defined as

$$M_{i,j} = \int_\Omega N_i\, N_j \,\mathrm{d}A \tag{1.24}$$

The load vector $\boldsymbol{b}$ is given by

$$b_i = \int_\Omega f\, N_i \,\mathrm{d}A$$

1.4 Helmholtz Equation

Helmholtz equation often arises in problems involving waves; in particular it is the time-independent version of the wave equation and is written as

$$\Delta u + k^2\, u = f \tag{1.25}$$

The solution u of Helmholtz equation represents the amplitude configuration of the wave in space, and k is the wavenumber.

It is easy now to see that the first term in (1.25) is the Laplacian operator we already encountered in the Poisson equation, while the second term is, besides the constant k, the same as in the least-squares fitting problem. Applying the same procedure as in the previous cases we are therefore able to rewrite the problem as searching for solutions of the linear system of equations

$$A\boldsymbol{u} + k^2 M\boldsymbol{u} = \boldsymbol{b} \;\Rightarrow\; \left(A + k^2 M\right)\boldsymbol{u} = \boldsymbol{b}$$

As we have seen with the above examples, the matrices A and M play an important role in the solution of partial differential equations using a Galerkin approach. Simple elliptic problems may use only the stiffness matrix A; simple geometrical problems may use only the mass matrix M; while more complex or time-dependant problems use both.

The space $\mathcal{V}_h$ can be defined using several different types of basis functions. This choice is of great importance as it will dictate the properties of the solution space. Different types of basis functions will yield different system matrices and consequently this will affect the convergence speed of numerical methods. For this reason, we will look at the impact the different basis functions presented in section 3 have on some important numerical properties of such matrices: conditioning number, bandwidth, and sparsity.

2

CAD Geometry

In this chapter, which is an excerpt from [22], we will synthetically present the main elements of the geometry used in CAD graphics. A more in-depth dissertation can be found in [8]. We will start by describing Bezier Curves, which are at the foundation of most of modern Computer Aided Design software. Since their introduction in 1966 by Pierre Bezier, they have completely shaped modern computer graphic thanks to their flexibility and simplicity. Bezier Curves provide an intuitive way to modify the shape of the curve, done through the use of control points.

Next, we will present the B-spline Curves following the approach given in [4]. B-spline curves are a generalization of Bezier curves which allow to solve some of their problems. We will also present in this chapter the three standard techniques for *global* refinement that B-splines naturally possess, namely the h-refinement, the p-refinement and the k-refinement. The techniques for *local* refinement, that will constitute the subject of our analysis, are presented in Chapter 3.

2.1 Bezier Curves

Bezier Curves are functions $P(t) : [0, 1] \longrightarrow \mathbb{R}^n$. They are piecewise polynomial parametric curves and are completely defined by the *Control Points*. The control points are an ordered sequence $\{P_0 \dots P_k\}$ of $k+1$ points, where each $P_i \in \mathbb{R}^n$ and k is the polynomial degree of the curve. They are the most important part in the definition of a Bezier curve since they define the relationship between the parametric space and $\mathbb{R}^n$, and are used to actively modify the geometry of the curve. This means that once the control points are chosen, the shape of the curve is completely defined by their position. The so called *Control Poligon* is obtained by linear interpolation of successive control points. We will also denote with $P_i^k(t)$ the curve of degree k originating from the control point P_i.

2.1.1 de Casteljau Algorithm

Given the polynomial degree and the control polygon, we can define the curve by a recursion on k as follows:

$\underline{k = 1}$

Consider all couples of successive control points $\{P_i, P_{i+1}\}$, $i = 0 \dots k-1$. Each couple defines a curve P_i^1, which is the straight line segment from P_i to P_{i+1},

given by

$$P_i^1(t) = (1-t)P_i + t\,P_{i+1}$$

$\underline{k=2}$

The curve of degree 2 is defined by linear interpolation of the curves of degree 1 defined by two successive couples of control points.

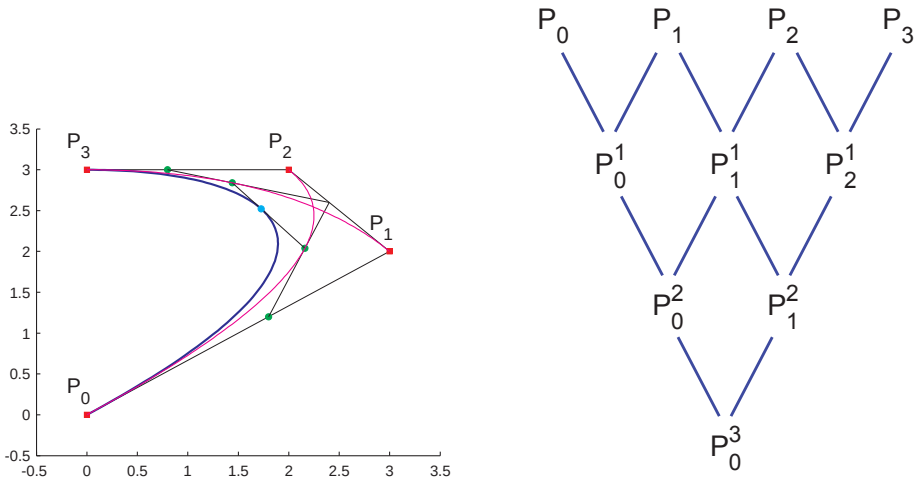
$$P_i^2(t) = (1-t)P_i^1(t) + t\,P_{i+1}^1(t)$$

This means that, given $\bar{t} \in [0,1]$, the segment from $P_i^1(\bar{t})$ to $P_{i+1}^1(\bar{t})$ is tangent to the quadratic curve in the point $P_i^2(\bar{t})$.

We can generalize this last observation, stating that *a Bezier curve of degree k is given by the envelope of the segments passing through the points of the curves of degree $k-1$ evaluated with the same parameter value.* Applying this procedure we get that the curve of degree k defined by the control points $\{P_0 \dots P_k\}$ is given by

$$P_0^k(t) = (1-t)P_0^{k-1} + t\,P_1^{k-1} \tag{2.1}$$

Figure 2.1 illustrates the de Casteljau's Algorithm for a curve of degree 3.



(a) **(b)**

Figure 2.1: **(a)** In blue, the cubic Bezier curve defined by $P_0 = [0,0], P_1 = [3,2], P_2 = [2,3], P_3 = [0,3]$, in pink the respective quadratic curves. The segments are drawn for $\bar{t} = 0.6$. **(b)** Diagram showing the various steps of the algorithm.

The construction of a Bezier curve through the de Casteljau's algorithm allows to recognise two important properties of these curves: the first is that the support of a Bezier curve is completely contained in the convex hull[1] defined by the control polygon. This is due to the fact that at each step of the algorithm, the points we get are lying on segments connecting curves with a lower polynomial degree. The second important property is

[1]The convex hull is defined by picking two consecutive control points, and considering the line passing through them; if all other control points are lying in the same semiplane respect to that line, then the segment between those two points is an edge of the convex hull, else we discard that line. This procedure is repeated for every couple of consecutive points; the collection of segments obtained in this way forms the convex hull, which is the smallest convex set containing all control points.

called *subdivision property*: given a parameter value $\bar{t}$, the point $P_0^k(\bar{t})$ subdivides the curve in two arches, and these arches are again Bezier curves. In Figure 2.1, the light-blue point subdivides the curve in two arches, each of which is a Bezier curve defined by the green control points. As a consequence, the original Bezier curve is contained in the union of the two convex hulls pertaining to the two arches, which is much smaller than the original convex hull. This feature is widely used in CAD geometry to determine possible intersections between curves.

2.1.2 Bernstein's Polynomials basis

Let's now take, for example, a cubic Bezier curve defined as in (2.1). We can expand the lower-degree curves recursively and reach an expression based only on the control points:

$$P_0^3(t) = (1-t)^3 P_0 + 3\,(1-t)^2\,t\,P_1 + 3\,(1-t)\,t^2 P_2 + t^3 P_3$$

We note that the control point's coefficients are given by the expansion of $((1-t)+t)^3$ and therefore their sum is always equal to 1 (this is another of they key property of Bezier curves). We can exploit this to derive a more compact notation for the curve.

Definition 3. Given the degree k of the curve, we can define the *Bernstein's Polynomials* as

$$B_i^k(t) = \binom{k}{i}(1-t)^{k-i}\,t^{\,i} \tag{2.2}$$

The $k+1$ Bernstein's polynomials $B_0^k, \ldots, B_k^k$ form a basis for the space of Bezier curves of degree k.

This is not the only choice available: we could also use different basis, like the standard base for polynomials spaces, but the reason of using Bernstein's polynomials reside in their numerical stability and the fact that, as we have seen, they are naturally associated to these kind of curves [8].

We now have a very compact expression for a Bezier's curve of degree k:

$$P_0^k(t) = \sum_{i=0}^{k} B_i^k(t) P_i \tag{2.3}$$

Figure 2.2 shows the Bernstein's polynomials creating the base of the cubic Bezier curves space.

We now list some of the main properties of Bezier curves:

Control of starting and ending point As we can see from Figure 2.2, generally a Bezier curve does not interpolate any of the control points, except the first and last one: evaluating the curve for $t = 0$ yields

$$P_0^k(0) = \sum_{i=0}^{k} B_i^k(0) P_i = P_0$$

and similarly when evaluating the curve for $t = 1$.

Invertibility It's easy to see that $B_i^k(t) = B_{k-i}^k(1-t)\,\forall i$. This means that the curves defined by the two control polygons $\{P_0, P_1 \ldots P_k\}$ and $\{P_k, P_{k-1} \ldots P_0\}$ have exactly the same support, but different (opposite) parametrization.

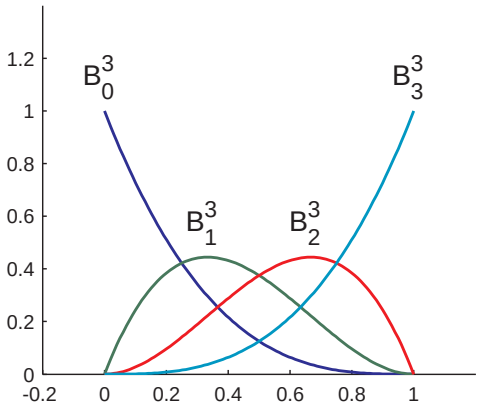


Figure 2.2: Bernstein's polynomials cubic base.

Partition of unity As we have seen, the Bernstein's polynomials are given by the expansion of terms like $((1-t)+t)^k$. This obviously means that

$$\sum_{i=0}^{k} B_i^k(t) = 1 \quad \forall\, t \in [0,1]$$

Therefore we say that the Bernstein's polynomials form a *partition of unity*. This is one of the most important features of Bezier curves and it is the key of their success in computer graphics: imagine we want to transform the curve with an affine transformation like $\bar{x} = Ax + c$. Applying the transformation directly to the curve yields:

$$\overline{P}(t) = AP(t) + c \overset{(2,3)}{\downarrow} \sum_{i=0}^{k} B_i^k(t) AP_i + c$$

If instead we apply the affine transformation to the control points, i.e. $P_i \longmapsto AP_i + c$, we get

$$\overline{P}(t) = \sum_{i=0}^{k} B_i^k(t)(AP_i + c) = \sum_{i=0}^{k} B_i^k(t) AP_i + \underbrace{\sum_{i=0}^{k} B_i^k(t)\, c}_{1}$$

Which is exactly the same result we had before. If we want to modify a Bezier curve with an affine transformation it's therefore sufficient to modify only the control points, and this allows us to avoid the high computational cost of applying the transformation to the full curve support.

Convex Hull As we pointed out before, a Bezier curve is fully contained in the convex hull defined by its control points. This is because at each step of the de Casteljau's algorithm the evaluated curve points lie on segments that connect lower-order curves and the lowest possible order is always 1, which refers to the straight lines between the control points.

2.1.3 Degree Elevation

An important feature of Bezier curves is that it is possible to increase the polynomial degree of the curve while maintaining its shape. From a geometric point of view, this is

useful when we want to "glue together" curves with different polynomial degree in order to achieve high regularity in the connection point. From a numerical-analysis point of view, this means that we can enrich the basis functions space without altering the geometry of the curve.

Say we have a curve defined by the control points $\{P_0, \dots P_k\}$ and we want to increase the polynomial degree from k to $k+1$. This means that we will have to define a new set of control points $\{Q_0, \dots Q_{k+1}\}$ such that

$$\sum_{i=0}^{k} B_i^k(t) P_i = \sum_{j=0}^{k+1} B_j^{k+1}(t) Q_j \qquad \forall\, t$$

To achieve this, we multiply the right hand side by $(1 - t + t)$ and get:

$$\begin{aligned}\left(\sum_{i=0}^{k} B_i^k(t) P_i\right)(1-t+t) = &\sum_{i=0}^{k} \binom{k}{i} (1-t)^{k+1-i}\, t^i P_i + \\ &\sum_{i=0}^{k} \binom{k}{i} (1-t)^{k-i}\, t^{i+1} P_i \qquad (2.4)\\ \text{And we want this to be } = &\sum_{j=0}^{k+1} \binom{k}{j} (1-t)^{k+1-j}\, t^j Q_j\end{aligned}$$

Now we need to set the coefficients of the same order terms to be equal:

$$\binom{k}{j} Q_j = \binom{k}{j} P_j + \binom{k}{j-1} P_{j-1}$$

Multiplying both sides by $\dfrac{(k-j)!\,(j-1)!}{k!}$ we have:

$$\frac{k+1}{(k+1-j)\,j}\, Q_j = \frac{1}{j}\, P_j + \frac{1}{k+1-j}\, P_{j-1}$$

and from this we finally get

$$Q_j = \left(1 - \frac{j}{k+1}\right) P_j + \left(\frac{j}{k+1}\right) P_{j-1} \qquad (2.5)$$

Equation (2.5) gives us the new set of control points that defines a curve of degree $k+1$ but with the same shape as the old curve. Figure 2.3 shows this procedure, used to increase the curve's degree from 3 to 4.

2.1.4 Problems

As we have seen, the Bezier curves have many good and interesting properties. They have, however, two main problems: first, there is no local control on the shape of the curve. If we move even one of the control points, in fact, all the curve will be modified. Second, they are not optimal in describing complicated geometries that will require a curve with a very high polynomial degree and, therefore, many control points and computational effort. An example of this is figure 2.4: to create the spiral we needed a Bezier curve of degree 29.

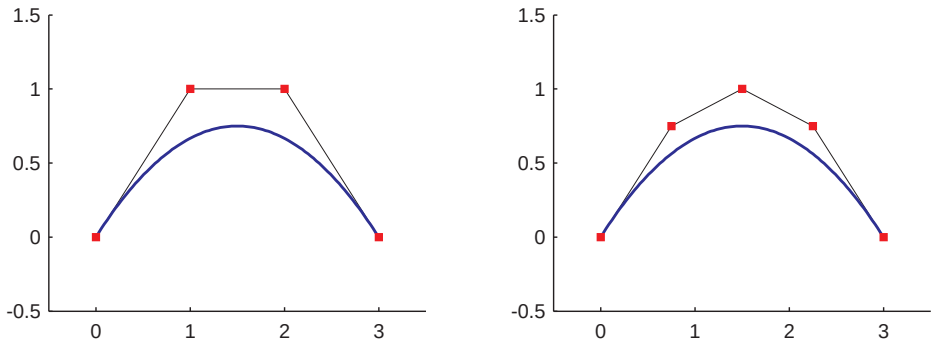


Figure 2.3: **(a)** Curve of degree 3 defined by four control points. **(b)** The new curve of degree 4. The new control points are calculated using equation (2.5).

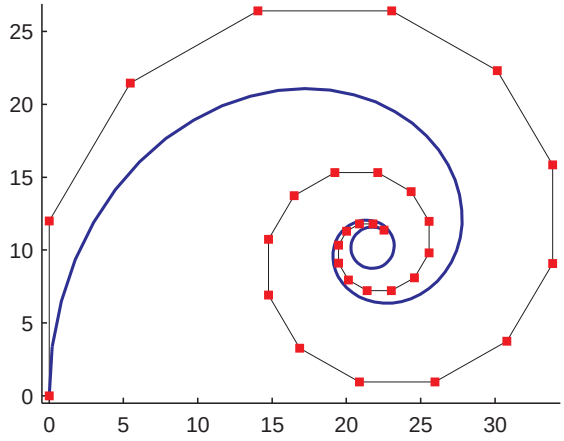


Figure 2.4: A spiral arc. To obtain this we used 30 control points.

2.2 B-Spline Curves

B-Spline curves can be seen as a generalization of the Bezier curves, of which they attempt to fix some of the problems. More specifically, B-Spline curves allows for local deformation of the curves and they do not need many control points in order to achieve a high polynomial degree.

2.2.1 The Knots Vector and the Basis Functions

As for the Bezier curves, B-Splines also require a set of control points $\{B_1 \dots B_n\}$, but one great advantage is that now the polynomial degree p of the curve is no longer fixed by the number of control points used.

Instead, the polynomial degree and the number of control points relate to the so-called knot vector in the following way: given n control points and the polynomial degree p of the curve, the *Knots Vector* is a non-decreasing sequence of coordinates in the parameter space of form $\Xi = \{\xi_1, \xi_2 \dots \xi_{n+p+1}\}$ where $\xi_i \in \mathbb{R}$ is a knot. The knot vector is said to be *uniform* if the knots are equally spaced, *non-uniform* otherwise.

An extremely important feature is the possibility to "collapse" some knots to the same value, i.e. to have repeated values in the knots vector. If the first and last knot value

have multiplicity $p+1$ then the knot vector is said to be *open*.

In the following, in order to ease notation, a knot with multiplicity m_i will be written as $\xi_i^{m_i}$. It is clear that we do not intend a power elevation.

As we will see, the multiplicity of a knot determines crucial properties of the basis functions and therefore affects directly the shape of the curve. In particular, the regularity of the basis functions decreases as the multiplicity of the knot increases.

Once we have chosen the knot vector Ξ, the polynomial degree p and the number n of the control points we intend to use, the functions $\{N_{1,p} \dots N_{n,p}\}$ form a base for the B-Spline curves space of degree p. These functions are defined recursively on p as follows:

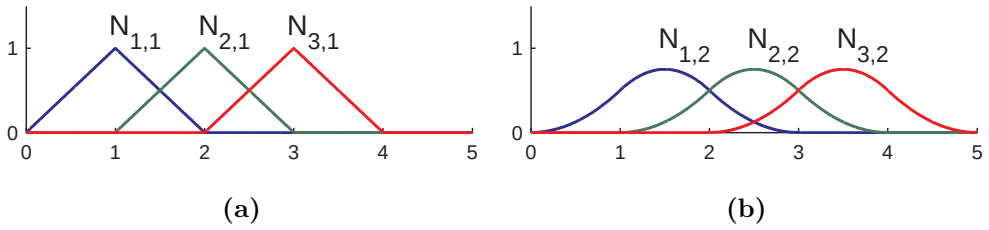
$p = 0$:

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leqslant \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$p > 0$:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.7}$$

This procedure in knows as the *de Boor-Cox Recursion*. Figure 2.5 shows the linear and quadratic base functions defined on the uniform knot vector $\Xi = \{0, 1, 2, 3, 4, 5\}$.



(a) **(b)**

Figure 2.5: **(a)** Basis functions of degree 1 defined on the knot vector $\{0, 1, 2, 3, 4, 5\}$. **(b)** Basis functions of degree 2 defined on the same knot vector.

We now make several remarks on the properties of these basis functions.

First, it's interesting to note that the the basis functions of degree $p = 0$ (the constants defined in (2.6)) and $p = 1$ are exactly the same as those used in classical Finite Elements. Quadratic basis functions, on the other hand, differ from the classical FEM ones: instead of distinguishing between the central and edge nodes, they are all the same and simply translated on the knot vector. This behaviour is maintained also for higher polynomial degrees.

As we can see, another feature is that for uniform, non-open knot vectors the functions does not form a partition of unity on the full parametric space, but only on a restricted portion of it. Specifically we have

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1 \qquad \xi \in [\xi_p, \xi_n] \tag{2.8}$$

This is a typical feature when working with uniform know vector. We will discuss this in deeper detail in the next sections.

Another major feature to note is that each basis function of degree p have $p-1$ continuous derivatives across element boundaries, i.e. across knots. This property is one of the main distinctive features of Isogeometric Analysis and have extremely important implications from a numerical point of view.

Another thing worth mentioning is that these functions are always non-negative on the entire parameter space, quite a difference compared to the classical finite element functions as this implies that the entries in the mass matrix will always be positive.

Lastly, each function of degree p acts on $p+2$ knot values (or $p+1$ knot spans), therefore high-order basis functions will be non-zero over much larger portions of the domain compared to the classical finite element functions. However, this feature does not influence the bandwidth of the system matrices as each basis function shares support with only $2p$ different functions, exactly the same as in the classical finite element case [4, p. 22].

We underline, for sake of clarity, that the basis functions are of degree p in a "piecewise" sense: from one knot value to the next one, while they will have regularity $p - m_i$ across a knot value of multiplicity m_i, as we will see in the next section.

2.2.2 Base Regularity on non-uniform open knot vectors

Open knot vectors are standard in CAD graphics. In 1D the functions base defined on an open knot vector is interpolatory[2] in the first and last knot, but normally it will not interpolate the internal knots. This is one of the differences between B-Spline basis and Finite Element lagrangian basis. Another consequence of using open knot vectors is that the edge of a B-spline object in dimension d is itself a B-spline object in dimension $d-1$. For example, the edge of a B-spline surface ($d=2$) is a B-spline curve ($d=1$).

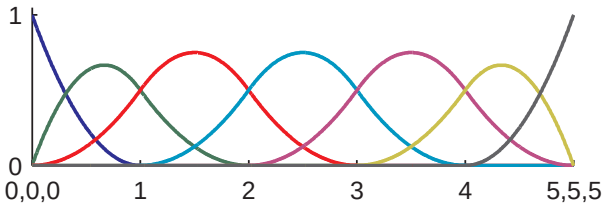


Figure 2.6: Quadratic B-Spline basis on the open knot vector $\Xi = [0^3, 1, 2, 3, 4, 5^3]$. The basis is interpolatory the first and last knot values.

Non-uniform vectors are particularly useful when we need to describe curves with low-regularity points, such as angular points or cusps (think, for instance, to a car's chassis). This diminished regularity is achieved repeating knot values.

Figure 2.7 shows the functions of the cubic B-Spline base defined on the knot vector $\Xi = [0^4, 1, 2, 3^3, 4, 5^2, 6^4]$. The base is C^{p-m_i} across knot values, where m_i is the multiplicity of that knot value and p is the polynomial degree of the functions. When the multiplicity equals the degree of the base, as for $\xi = 3$, the base is C^0 and it's interpolatory. When the multiplicity is $p+1$, as for the first and last knot values, the base is "C^{-1}", meaning that it becomes discontinuous, therefore creating the curves edges. The knot value $\xi = 5$ have a multiplicity of 2, which means that the base is C^1 across that knot. In all other knot values the regularity is the maximum possible, namely C^2.

[2]The base, by itself, does not interpolate anything of course. What we mean is: when a knot have multiplicity equal or greater than the curve's degree, all basis functions will vanish at that point, except one that will take value 1, hence the curve will interpolate the corresponding control point.

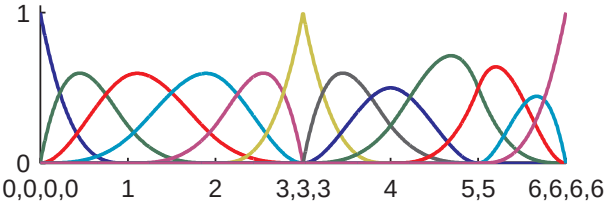


Figure 2.7: Cubic B-Spline basis on the open knot vector $\Xi = [0^4, 1, 2, 3^3, 4, 5^2, 6^4]$. The basis' regularity is strictly connected to knot's multiplicity.

2.2.3 Derivatives of B-spline basis functions

Thanks to the recursive definition given in (2.6)-(2.7), the derivatives of the basis functions can also be expressed in terms of lower-order basis functions.

Given a base of degree p defined on the knot vector Ξ, the derivative of the i-th basis function is given by

$$\frac{\mathrm{d}}{\mathrm{d}\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.9}$$

Proof. It's useful to explicitly write the basis functions involved in the derivative formula: from (2.7) we have

$$\begin{aligned} N_{i,p-1} &= \underbrace{\frac{\xi - \xi_i}{\xi_{i+p-1} - \xi_i} N_{i,p-2}}_{A} + \underbrace{\frac{\xi_{i+p} - \xi}{\xi_{i+p} - \xi_{i+1}} N_{i+1,p-2}}_{B} \\ N_{i+1,p-1} &= \underbrace{\frac{\xi - \xi_{i+1}}{\xi_{i+p} - \xi_{i+1}} N_{i+1,p-2}}_{C} + \underbrace{\frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+2}} N_{i+2,p-2}}_{D} \end{aligned} \tag{2.10}$$

We will prove the formula by induction on p:

$\underline{p = 0}$

In this case we have

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leqslant \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Therefore the derivative is always null, in accordance with (2.9) for $p = 0$.

$\underline{p-1 \rightsquigarrow p}$

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\xi} N_{i,p}(\xi) &= \frac{\mathrm{d}}{\mathrm{d}\xi} \left[\frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1} + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p+1} \right] \\ &= \frac{1}{\xi_{i+p} - \xi_i} N_{i,p-1} + \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} \frac{\mathrm{d}}{\mathrm{d}\xi} N_{i,p-1} \\ &\quad - \frac{1}{\xi_{i+p+1} - \xi_i} N_{i+1,p-1} + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} \frac{\mathrm{d}}{\mathrm{d}\xi} N_{i+1,p-1} \end{aligned}$$

We now apply the inductive hypothesis to the derivatives of the functions of degree $p-1$, in this way we have:

$$
\begin{aligned}
&= \frac{1}{\xi_{i+p} - \xi_i} N_{i,\,p-1} \\
&\quad + \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} \left[\frac{p-1}{\xi_{i+p-1} - \xi_i} N_{i,\,p-2} - \frac{p-1}{\xi_{i+p} - \xi_{i+1}} N_{i+1,\,p-2} \right] \\
&\quad - \frac{1}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,\,p-1} \\
&\quad + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} \left[\frac{p-1}{\xi_{i+p} - \xi_{i+1}} N_{i+1,\,p-2} - \frac{p-1}{\xi_{i+p+1} - \xi_{i+2}} N_{i+2,\,p-2} \right]
\end{aligned}
$$

We now do the multiplications ordering the numerators and denominators in order to form some pieces of (2.10):

$$
\begin{aligned}
&= \frac{1}{\xi_{i+p} - \xi_i} N_{i,\,p-1} + \frac{p-1}{\xi_{i+p} - \xi_i} \cdot \underbrace{\frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,\,p-2}}_{A} \\
&\quad - \frac{p-1}{\xi_{i+p} - \xi_i} \cdot \frac{\xi - \xi_i}{\xi_{i+p} - \xi_{i+1}} N_{i+1,\,p-2} \\
&\quad - \frac{1}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,\,p-1} - \frac{p-1}{\xi_{i+p+1} - \xi_{i+1}} \cdot \underbrace{\frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+2}} N_{i+2,\,p-2}}_{D} \\
&\quad + \frac{p-1}{\xi_{i+p+1} - \xi_{i+1}} \cdot \frac{\xi_{i+p+1} - \xi}{\xi_{i+p} - \xi_{i+1}} N_{i+1,\,p-2}
\end{aligned}
$$

and we then sum the remaining coefficients:

$$
\begin{aligned}
&= \frac{1}{\xi_{i+p} - \xi_i} N_{i,\,p-1} + \frac{p-1}{\xi_{i+p} - \xi_i} \cdot A \\
&\quad + \left[\frac{p-1}{\xi_{i+p+1} - \xi_{i+1}} \cdot \frac{\xi_{i+p+1} - \xi}{\xi_{i+p} - \xi_{i+1}} - \frac{p-1}{\xi_{i+p} - \xi_i} \cdot \frac{\xi - \xi_i}{\xi_{i+p} - \xi_{i+1}} \right] N_{i+1,\,p-2} \\
&\quad - \frac{1}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,\,p-1} - \frac{p-1}{\xi_{i+p+1} - \xi_{i+1}} \cdot D = (*)
\end{aligned}
$$

We now have to verify that the terms in parenthesis can be written in the right way, in order to form the missing parts of (2.10); we therefore have to check that:

$$\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}\cdot\frac{\xi_{i+p+1}-\xi}{\xi_{i+p}-\xi_{i+1}}-\frac{p-1}{\xi_{i+p}-\xi_i}\cdot\frac{\xi-\xi_i}{\xi_{i+p}-\xi_{i+1}}$$

$$||\ ?$$

$$\underbrace{\frac{p-1}{\xi_{i+p}-\xi_i}}_{\substack{\text{coefficient}\\ \text{of } A}}\cdot\underbrace{\frac{\xi_{i+p}-\xi}{\xi_{i+p}-\xi_{i+1}}}_{\substack{\text{missing part}\\ \text{of } B}}-\underbrace{\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}}_{\substack{\text{coefficient}\\ \text{of } D}}\cdot\underbrace{\frac{\xi-\xi_{i+1}}{\xi_{i+p}-\xi_{i+1}}}_{\substack{\text{missing part}\\ \text{of } C}}$$

The equality indeed holds, its sufficient to sum all the terms with same coefficient to have:

$$\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}\left[\frac{\xi_{i+p+1}-\xi}{\xi_{i+p}-\xi_{i+1}}+\frac{\xi-\xi_{i+1}}{\xi_{i+p}-\xi_{i+1}}\right]=\frac{p-1}{\xi_{i+p}-\xi_i}\left[\frac{\xi_{i+p}-\xi}{\xi_{i+p}-\xi_{i+1}}+\frac{\xi-\xi_i}{\xi_{i+p}-\xi_{i+1}}\right]$$

$$\frac{1}{\xi_{i+p+1}-\xi_{i+1}}\left[\frac{\xi_{i+p+1}-\xi_{i+1}}{\xi_{i+p}-\xi_{i+1}}\right]=\frac{1}{\xi_{i+p}-\xi_i}\left[\frac{\xi_{i+p}-\xi_i}{\xi_{i+p}-\xi_{i+1}}\right]$$

$$\frac{1}{\xi_{i+p}-\xi_{i+1}}=\frac{1}{\xi_{i+p}-\xi_{i+1}}$$

We can therefore rewrite the terms in the square brackets of our formula to obtain:

$$\begin{aligned}(*)=&\frac{1}{\xi_{i+p}-\xi_i}N_{i,\,p-1}+\frac{p-1}{\xi_{i+p}-\xi_i}\cdot A\\&+\left[\frac{p-1}{\xi_{i+p}-\xi_i}\underbrace{\frac{\xi_{i+p}-\xi}{\xi_{i+p}-\xi i+1}}_{\text{coeff. of } B}-\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}\underbrace{\frac{\xi-\xi_{i+1}}{\xi_{i+p}-\xi_{i+1}}}_{\text{coeff. of } C}\right]N_{i+1,\,p-2}\\&-\frac{1}{\xi_{i+p+1}-\xi_{i+1}}N_{i+1,\,p-1}-\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}\cdot D\\[2ex]=&\frac{1}{\xi_{i+p}-\xi_i}N_{i,\,p-1}+\frac{p-1}{\xi_{i+p}-\xi_i}[A+B]\\&-\frac{1}{\xi_{i+p+1}-\xi_{i+1}}N_{i+1,\,p-1}-\frac{p-1}{\xi_{i+p+1}-\xi_{i+1}}[C+D]\\[2ex]=&\frac{p}{\xi_{i+p}-\xi_i}N_{i,\,p-1}-\frac{p}{\xi_{i+p+1}-\xi_{i+1}}N_{i+1,\,p-1}\end{aligned}$$

■

Higher-order derivatives can be calculated further differentiating both terms in (2.9). The k-th order derivative can be generally expressed as

$$\frac{\mathrm{d}^{\mathrm{k}}}{\mathrm{d}\xi^k}N_{i,p}(\xi)=\frac{p\,!}{(p-k)\,!}\sum_{j=0}^{k}\alpha_{k,j}\,N_{i+j,p-k}(\xi)\tag{2.11}$$

where the coefficients $\alpha_{k,j}$ are given by

$$\alpha_{0,0} = 1$$
$$\alpha_{k,j} = \frac{\alpha_{k-1,j} - \alpha_{k-1,j-1}}{\xi_{i+p+1-k+j} - \xi_{i+j}} \qquad j = 1 \dots k-1$$

where it's assumed $\alpha_{a,b} = 0$ if $b > a$ or $b < 0$.

It's worth noting that the denominators of some coefficients can be zero when a knot value is repeated, in these cases the coefficient is defined to be equal to zero.

2.2.4 Curve and Mesh construction

Exactly as the Bezier curves, B-Splines can also be expressed as a linear combination of the basis functions with the control points.

As we noted before, the polynomial degree of the curve no longer depends on the number of control points, therefore in order to define a B-Spline curve we use the following procedure: we first need to define the n control points $\{B_1 \dots B_n\}$, where $B_i \in \mathbb{R}^d$. We then have to fix the polynomial degree p of the basis and assign the knot values $\{\xi_1 \dots \xi_m\}$ such that $m = n + p + 1$. We then apply (2.6) - (2.7) to get the basis functions $\{N_{1,p} \dots N_{n,p}\}$. The B-Spline curve is the linear combination of these functions with the control points:

$$C(\xi) = \sum_{i=1}^{n} N_{i,p} B_i \tag{2.12}$$

Clearly, the curve inherits most of the properties of the base, the most important being the regularity degree.

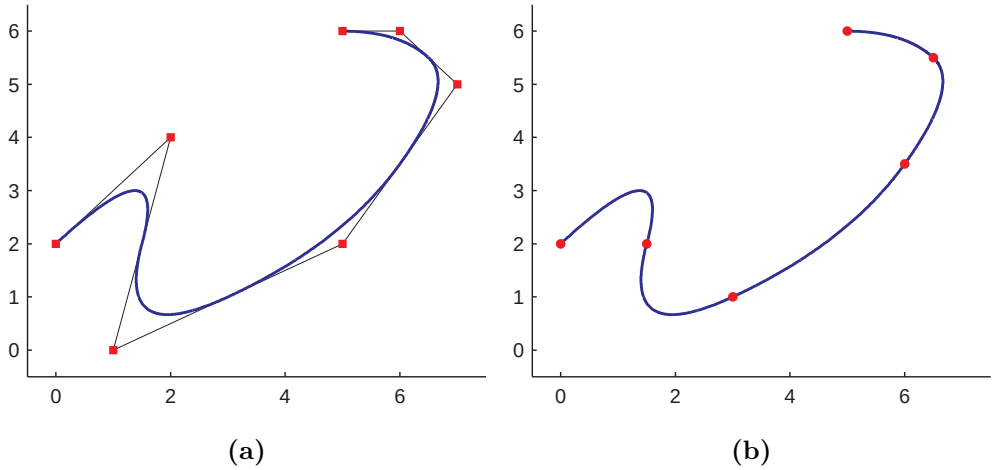


Figure 2.8: Example of piecewise quadratic B-Spline curve. **(a)** The curve and the control polygon. **(b)** The knot values images partition the curve into elements, generating the mesh. Basis functions and knot vector are the same as in fig. 2.6.

Figure 2.8 shows a quadratic B-Spline curve defined using the same base as in Figure 2.6. In 2.8a it is shown the control polygon. As we can see the curve interpolates only the first and last control point. In Figure 2.8b we have plotted the images of the knot values. As we can see, the knot values partition the curve into the elements. This concept is crucial as it is the way meshes are generated in the Isogeometric setting: they are the

images of knot values under the geometrical mapping defined by the control points and the basis functions.

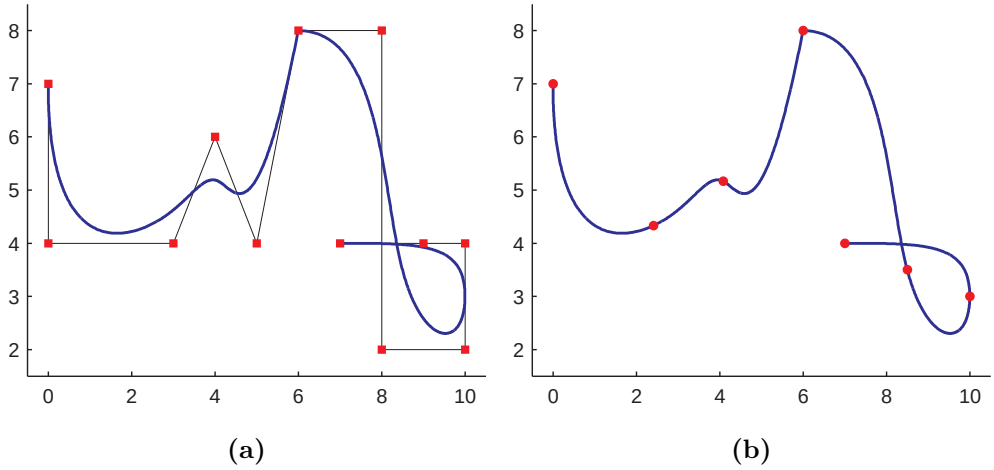


Figure 2.9: Example of piecewise cubic B-Spline curve.

A much more flexible behaviour is obtained using a non-uniform knot vector. Figure 2.9 shows a cubic curve defined using the same basis functions as in Figure 2.7.

As we can see, the curve now interpolates the first and last control points as usual (given the open knot vector), but also interpolates the sixth control point, corresponding to the repeated knot value $\xi = 3$, whose multiplicity equals the polynomial degree of the curve. The curve, moreover, is clearly C^0 in that point. We also note that the curve is tangent to the control polygon in correspondence of the repeated knot value $\xi = 5$. Again, we note that the images of the knot values partition the curve into elements, generating the mesh.

An important difference between the classic finite element meshes and the B-Spline meshes we have illustrated here is that in the latter case the parametric space extends over the full object and not over the single element only. In fact, in classic finite elements methods the parametric space (or reference element) is mapped to each physical element by a different mapping. On the opposite, using B-Splines we have only one mapping that "covers" the full physical domain. The parameter space is partitioned in intervals by the knot values and each interval corresponds to an element in the physical domain. This situation is illustrated in figure 2.10 taken from [4].

2.2.5 Mesh and Basis refinement

B-Splines geometries offer several different techniques to refine the mesh or the function spaces: the control over the basis' regularity together with the possibility of inserting new knot values or increasing the polynomials degree offer a wider array of choices compared to the classical finite element framework. We present here the three standard refinement techniques: knot insertion (h-refinement), degree elevation (p-refinement) and the combination of the two (k-refinement). Please note that these techniques exploit the natural tensor-product structure of B-spines in dimensions higher than 1. Due to this, if we for example insert one knot value in one parametric direction, the meshline generated by that knot will extend through the full mesh. This means that these techniques, while very useful to understand the concept of refinement in the B-spline setting, are applicable

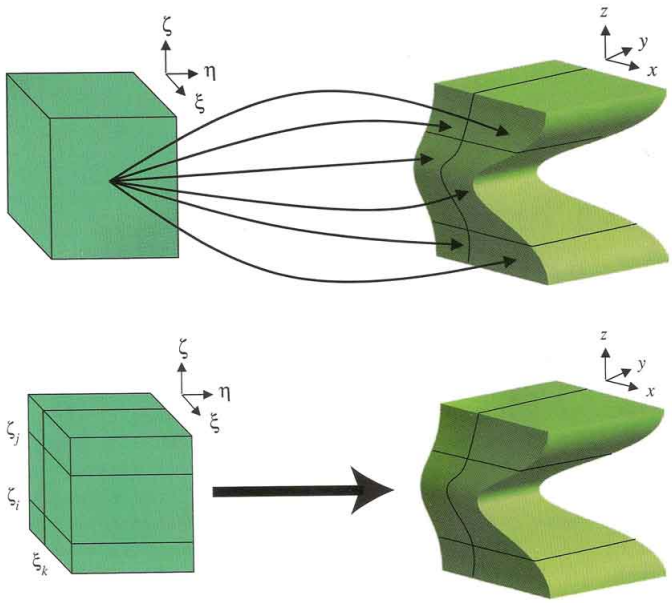


Figure 2.10: Up: In classical finite elements methods, the reference element is mapped to each physical element via a different mapping. **Down:** Using B-Splines only one mapping is necessary. Elements boundaries are generated by the knot values.

only to perform a *global* refinement. A *local* refinement is also possible, and in many cases desirable. We will present several such techniques in Chapter 3.

2.2.5.1 Knot insertion

The first technique to refine the mesh consists of inserting new knot values while keeping the basis' order fixed. This technique can be considered the equivalent of the FEM h-refinement. We present here the process of insertion for one new knot value, the ideas however can be further generalized to the multi-insertion case.

The aim is to extend the knot vector, going from $\Xi = \{\xi_1 \dots \xi_{n+p+1}\}$ to the new vector $\overline{\Xi} = \{\xi_1 \dots \xi_k, \bar{\xi}, \xi_{k+1} \dots \xi_{n+p+1}\}$, $\bar{\xi}$ being the new knot value. The new basis functions are constructed using (2.6) - (2.7) with the new knot vector $\overline{\Xi}$.

Once the new knot values are defined, we need to update the control polygon as well, in order to maintain the curve's geometry. In analogy with what we have already seen for the curves, the new control polygon $\{\overline{B}_1 \dots \overline{B}_{n+1}\}$ is defined with a linear combination of the old n control points:

$$\overline{B}_i = \alpha_i B_i + (1 - \alpha_i) B_{i-1} \quad i = 1 \dots n+1 \tag{2.13}$$

where we now have

$$\alpha_i = \begin{cases} 1 & 1 \leqslant i \leqslant k-p \\ \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k-p+1 \leqslant i \leqslant k \\ 0 & k+1 \leqslant i \leqslant n+1 \end{cases} \tag{2.14}$$

In order to insert more then one new knot value it's sufficient to repeat the application of (2.13) - (2.14).

It's interesting to note that with this technique it's possible to increase the multiplicity of a knot value; as we have seen this will affect the regularity of the basis functions but, thanks to how the new control points are set, the continuity of the curve will remain unchanged.

The Knot insertion have clear analogies with the finite elements' h-refinement, as it allows to subdivide elements into smaller ones; it differs, however, in the number of new basis functions that are introduced (here we have only one new function for each new unique knot value) and in the regularity of basis functions across element boundaries. On the other hand, he possibility of increasing the multiplicity of existing knot values have no counterpart in classical finite element analysis.

2.2.5.2 Degree elevation

Another refinement technique consist in increasing the polynomial degree of the basis functions without introducing new (unique) knot values. This process can be seen as the isogeometric equivalent of p-refinement.

It is important to note however that an increase in the basis' degree must be associated with a corresponding increase in each knot's multiplicity, in order to maintain the basis' regularity across knot values.

The procedure is as follows: the B-Spline curve is first subdivided into many Bezier curves, using the knot-insertion procedure to increase the multiplicity of all internal knot values until it equals the polynomial degree of the curve. At this point it's possible to increase the polynomial degree of each Bezier curve we obtained, as explained in section 2.1.3, doing so the new control points are generated. Once this is done we can remove the extra knot values until we have a knot vector that is exactly the same as the original one, but with all values' multiplicity increased by one (or more, if the polynomial degree has been increased more then by one). At the end of this procedure we will have obtained a new contro polygon and a new knot vector that can be used to define the new basis functions.

This technique have much in common with the classical p-refinement but the main difference is that it can be applied on basis functions with any kind of regularity, while the original p-refinement requires the basis to be C^0. Of course, if the original B-Spline curve is already C^0 across knot values, the two techniques are exactly the same.

2.2.5.3 *k*-refinement

The greater flexibility of the degree elevation technique for B-Splines permits to combine the procedure with knots insertions leading to a refinement technique that have no counterpart in classical finite elements: the k-refinement.

This technique exploit the fact that knot insertion and degree elevation procedures are non-commutative: if a new, unique knot value $\bar{\xi}$ is inserted in a curve of polynomial degree p, the new resulting base will have $p-1$ continuous derivatives across that knot value. If we now increase the curve's degree to q, the new base will still have only $p-1$ continuous derivatives across $\bar{\xi}$, despite being now of degree q. This is because during degree elevation the multiplicity of each knot is increased in oder to maintain the basis' regularity.

On the opposite, if we *first* elevate the curve to degree q and *then* we insert the new knot value $\bar{\xi}$, the new resulting base will now have $q-1$ continuous derivatives. This latter procedure is referred to as k-refinement.

k-refinement is potentially a superior approach to high-precision analysis compared to classical h- or p-refinement [4, p. 43-44].

2.2.6 B-Splines and Bezier curves

We now comment a little more in detail the connection between B-Spline curves and Bezier curves, and we do this with an example: consider a cubic B-Spline curve defined with four control points B_1, B_2, B_3, B_4, on the knot vector $\Xi = \{0, 0, 0, 0, 1, 1, 1, 1\}$. This knot vector allows us to have a curve parametrized on $[0, 1]$ exactly as the Bezier curves.

Using (2.6) - (2.7) it's easy to verify that the basis functions are

$$N_{1,3} = (1-\xi)^3 \qquad N_{2,3} = 3\,\xi\,(1-\xi)^2$$
$$N_{3,3} = 3\,\xi^2(1-\xi) \quad N_{4,3} = \xi^3$$

which are exactly the Bernstein's polynomials of degree 3 defined in (2.2).

2.2.7 Higher Dimensions: Tensor Product Basis

A very convenient feature of CAD geometries is that multi-dimensional basis functions are obtained from the 1D basis functions via tensor-product.

To define a B-Spline surface we will therefore need a set of points $\{B_{i,j}\}$ where $i = 1 \dots n$, $j = 1 \dots m$, called *control net*. Assigned the polynomials degrees p and q corresponding to the two physical dimensions and the relative knot vectors $\Xi = \{\xi_1 \dots \xi_{n+p+1}\}$ and $\mathcal{H} = \{\eta_1 \dots \eta_{m+q+1}\}$, the B-Spline surface is defined as

$$S(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi)\, M_{j,q}(\eta)\, B_{i,j} \tag{2.15}$$

where $N_{i,p}$ and $M_{j,q}$ are the one-dimensional basis functions defined on knot vectors Ξ and $\mathcal{H}$ respectively.

Many of the properties of a B-Spline surface follows directly from its definition using a tensor-product form, in particular the basis functions

$$\Phi_{i,j;\,p,q}(\xi, \eta) = N_{i,p}(\xi)\, M_{j,q}(\eta)$$

are a partition of unity on $[\xi_1, \xi_{n+p+1}] \times [\eta_1, \eta_{m+q+1}]$ and the regularity degree in one direction depends on the regularity of the corresponding base.

It's worth noting that the basis functions written like $\Phi_{i,j;\,p,q}$ use a *local* indexing. In many numerical applications it's useful to have also a *global* indexing of the form $\Phi_1, \Phi_2, \dots$. The local-to-global relation between indices is something that depends on both the situation we are facing and the programmer's preferences.

The same procedure can be generalized when we define 3D B-Spline objects: we will now need three polynomial degrees p, q and r, and their respective knot vectors Ξ, $\mathcal{H}$ and $\mathcal{Z}$; the B-Spline solid is then defined as

$$S(\xi, \eta, \zeta) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{l} N_{i,p}(\xi)\, M_{j,q}(\eta)\, L_{k,r}(\zeta)\, B_{i,j,k} \tag{2.16}$$

where the set of points $\{B_{i,j,k}\}$ is called *control lattice*.

3

Local Refinement techniques

In this chapter we present the theory of the three classes of spline functions we considered: Classical Hierarchical, Truncated Hierarchical and LR B-splines. An effort has been made in order to unify the different concepts under a common framework of notations, to ease both the understanding and the comparison of the different technologies. We have included only the essentials and we refer the interested reader to the papers on which we based our studies for an in-depth introduction and details [6, 11, 15, 28].

3.1 Notation and common definitions

The Hierarchical (both Classical and Truncated) and the LR B-splines methodologies use quite different points of view when considering meshes and refinements. As such, different notations have been developed in the corresponding publications. We will from now on use the following notation when we will need to address mesh-related quantities:

- ϵ for meshlines;
- Ω for domains, i.e. regions of the mesh (excluding mesh lines);
- V for full tensor product meshes;
- $\mathcal{M}$ for general meshes.

In particular, the Hierarchical setting focuses more on regions of the mesh and their underlying full tensor product meshes. For these reasons, Ω and V are often used in this context. The LR B-splines setting instead focuses more on meshlines and meshes as a whole. To provide a formal description of these different point of views we can write that a mesh $\mathcal{M}$ is seen as

$$\mathcal{M} = \bigcup_l (\Omega^l \cap V^l) \quad \text{in the Hierarchical setting}$$

$$\mathcal{M} = \bigcup_i \epsilon_i \quad \text{in the LR B-splines setting}$$

where the index l denotes the Hierarchical level and i runs over all meshlines.

The notation we will use for basis functions is the following:

- $N \in \mathcal{N}$ for *uniform* tensor product basis functions.

- $B \in \mathcal{B}$ for tensor product basis functions (possibly non-uniform).
- $H \in \mathcal{H}$ for Classical Hierarchical basis functions.
- $T \in \mathcal{T}$ for Truncated Hierarchical basis functions.
- $L \in \mathcal{L}$ for LR B-splines basis functions.

Of course, there exist cases where for some indices $N_i = B_i = H_i = T_i = L_i$, but we hope the different notation will ease the understanding of the technologies.

We have from elementary spline theory that a *knot vector* is a nondecreasing sequence of coordinates in the parameter space of the form $\Xi = [\xi_1, \xi_2, \ldots, \xi_{n+p+1}]$, where each $\xi_i \in \mathbb{R}$ is called a *knot*. If the knot values are equidistant the knot vector is called *uniform*, and *non-uniform* otherwise. If the first and last knots have multiplicity p+1, the knot vector is called *open*. A knot vector comprising of n+p+1 knot values will generate n univariate linearly independent basis functions of degree p. We will focus our analysis on B-splines built from uniform, non-open knot vectors. It is however important to remind that, with some additional work, is possible to generalize the same numerical tests using open or non-uniform knot vectors.

Definition 4. Given a knot vector $\Xi = [\xi_1, \xi_2, \ldots, \xi_{n+p+1}]$ and a polynomial degree p, the n univariate basis functions $B_{1,p}, \ldots, B_{n,p}$ are recursively defined in the following way:

$p = 0$:

$$B_{i,0}(\xi) = \begin{cases} 1 & \text{for } \xi_i \leqslant \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

$p > 0$:

$$B_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi) \tag{3.2}$$

The above definition is known as the *Cox-de Boor recursion formula*. From this definition it follows that each basis function depends only on p+2 knot values. For instance, for p=2 the knot vectors $\Xi = [0, 1, 2, 3, 4, 5]$ will generate three basis functions corresponding to the *local* knot vectors $\Xi_1 = [0, 1, 2, 3]$, $\Xi_2 = [1, 2, 3, 4]$, and $\Xi_3 = [2, 3, 4, 5]$. Due to this, we will often refer to the basis functions using their local knot vectors. The notation will also be adjusted on a case-to-case basis depending on what we need to emphasize, and we will use $B_i = B_{\Xi_i}$ to keep track of the local knot vector on which the function is built.

From the univariate basis functions it is possible to define multivariate functions using the tensor product structure of B-splines:

Definition 5. A n-variate B-spline $B(\boldsymbol{\xi})$ of degrees $\boldsymbol{p} = [p_1, p_2, \ldots, p_n]$ is a separable function $B : \mathbb{R}^n \longrightarrow \mathbb{R}$ defined as:

$$B_{\boldsymbol{\Xi}}(\boldsymbol{\xi}) = \prod_{i=1}^{n} B_{\Xi_i}(\xi_i)$$

where $\Xi_i \in \mathbb{R}^{p_i+1}$ is the local knot vector for the univariate basis function of degree p_i along the i-th parametric dimension.

Note that the polynomial degree is implicitly defined by the number of knots in the local knot vectors. In the bivariate setting, it is customary to denote the two parametric coordinates as ξ and η, and the corresponding polynomial orders as p and q. We denote a *tensor product basis* $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ as a basis of functions defined on a tensor product mesh.

In the following we will construct the mesh such that the actual domain will be the unit square, i.e. the initial tensor product basis will form a partition of unity on $\Omega_0 = [0,1] \times [0,1]$. Since we will use uniform knot vectors, we will need to extend the mesh beyond Ω_0 trough the use of a *ghost domain* G. In this way the full parametric domain will be given by $G \cup \Omega_0$.

We will represent bivariate basis functions on the same plot through the use of anchors. A common choice for the coordinates of the anchor are the *Greville abscissae*. The Greville abscissae $(\bar{\xi}, \bar{\eta})$ corresponding to a basis function are defined as

$$\bar{\xi} = \frac{1}{p} \sum_{j=2}^{p} \xi_j, \quad \bar{\eta} = \frac{1}{q} \sum_{j=2}^{q} \eta_j \tag{3.3}$$

where ξ_j and η_j are the knot values in the local knot vector. One drawback of Greville abscissae, however, is that they work nicely only if the function under consideration has a rectangular support, like normal B-splines. As we will see, it is very natural for Truncated Hierarchical B-splines to have non-rectangular support. In this case we will use *area-averaged coordinates* defined as

$$\bar{\xi} = \frac{\sum_{i:E_i \in \mathrm{supp}T} A_i \, \xi_i^c}{\sum_{i:E_i \in \mathrm{supp}T} A_i}, \quad \bar{\eta} = \frac{\sum_{i:E_i \in \mathrm{supp}T} A_i \, \eta_i^c}{\sum_{i:E_i \in \mathrm{supp}T} A_i} \tag{3.4}$$

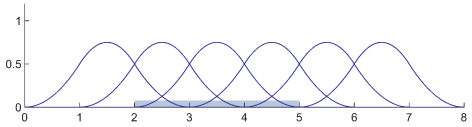
where the sum runs over all elements E_i in the support of the function, (ξ_i^c, η_i^c) are the coordinates of the centre of the element E_i and A_i is its area. The weighting by the area ensures that each element contributes in the right way to the resulting coordinate of the anchor. This method of calculating the coordinates returns the same result as the normal Greville abscissae in the case of rectangular support, while it allows to see the difference when the support is non-rectangular.

3.2 Hierarchical B-Splines

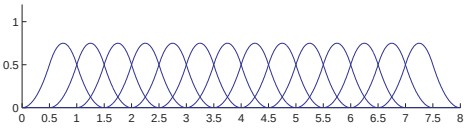
The application of the Hierarchical framework in Isogeometric Analysis is very well explained by Vuong et al. in [28], and Giannelli et al. in [11]. We will look at how an admissible mesh is constructed, and how the construction procedure defines a sequence of nested bounded domains linked to the different Hierarchical levels.

3.2.1 Introduction and general idea

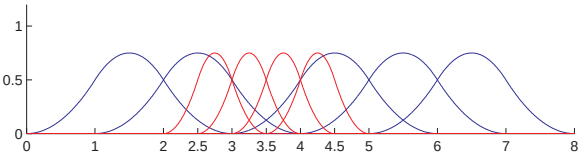
The basic idea underlying Hierarchical B-splines is very simple, yet results in a good and flexible method to locally refine the mesh. A one-dimensional example for quadratic basis functions is illustrated in Figure 3.1: one portion of the initial level 0 mesh is selected for refinement. The coarse basis functions contained in that area are substituted by finer basis functions, and we thus obtain the Hierarchical basis.



(a) The coarse mesh with the initial basis (level 0)



(b) The fine-scale basis (level 1)



(c) The Hierarchical basis, constructed as a suitable combination of the coarse and fine scale basis functions

Figure 3.1: Hierarchical Basis: Construction of a univariate basis using quadratic basis functions. The highlighted area is selected for refinement, and the coarse functions contained therein are substituted by finer basis functions.

As we can see from Figure 3.1 this basis does not constitute a partition of unity. This property is however easily recovered by weighting the functions appropriately (see Figure 3.2). In this case it is important to note that the region selected for refinement must contain at least the support of one coarse basis function. If this condition is not met, the weights associated with the finer basis functions will all be zero, and therefore no change will happen in the basis [28].

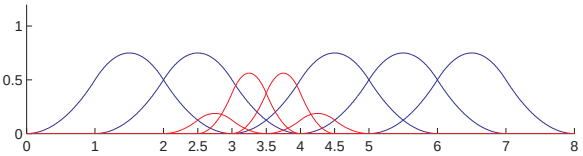


Figure 3.2: Hierarchical Basis: Weighting the fine scale functions appropriately ensures partition of unity for $\xi \in [2, 6]$.

Figure 3.3 presents the Hierarchical approach on a simple 2D example with biquadratic basis functions. When a selected area of the mesh is refined, the knot spans are halved in each direction and this introduces one new level in the hierarchy. The basis functions from the previous level that are contained in the refined region are then substituted by the corresponding finer basis functions defined on the new knot spans. As in the univariate case, an appropriate weighting of the functions can be used to recover the partition of unity.

In the following we will focus mainly on the two dimensional case, and several examples will be presented.

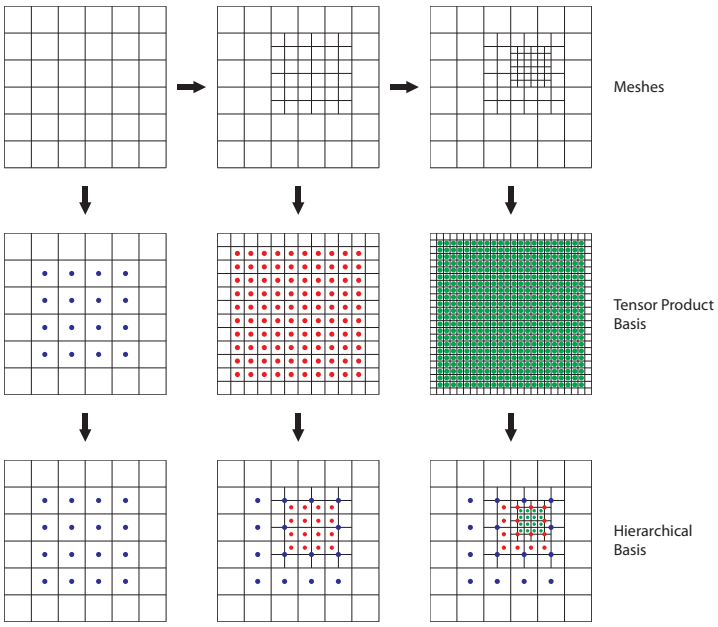


Figure 3.3: 2D Hierarchical Basis: Example using biquadratic basis functions. **Upper row**: A two-step refinement is applied to the initial mesh. **Middle row**: The tensor-product basis defined on the finest knot span available. From here we select the appropriate basis functions to include in the Hierarchical basis. **Lower row**: The actual Hierarchical basis defined on the refined mesh above. At each step, the basis functions from the previous level that are contained in the refined region are substituted by the finer ones. The anchors are positioned using Greville Abscissae.

3.2.2 The Classical Hierarchical Basis

The construction of the mesh on which the Hierarchical basis is defined is a direct application of the idea presented above: starting from an initial, uniform tensor product mesh V^0, some areas are selected for refinement. Once the areas have been selected, several new meshlines are introduced, halving the knot spans of the local knot vectors of all the functions contained therein. Of course, the selected areas must satisfy some rules in order to produce an admissible mesh. We will for now be general in order to understand the procedure, and we will state those rules in a few lines.

A mesh constructed in such a way defines a sequence of $M+1$ nested bounded domains

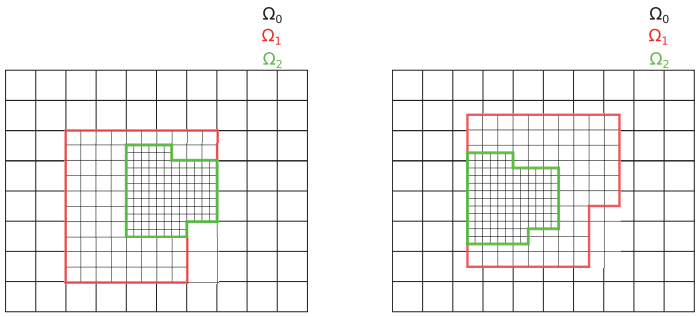
$$\Omega^M \subseteq \Omega^{M-1} \subseteq \ldots \subseteq \Omega^0$$

where M is the total number of new levels introduced in the mesh. To each level l is associated an underlying tensor product basis $\mathcal{N}^l$, defined on a tensor mesh V^l. In the following we will consider the support of each function as restricted to the domain Ω^0, i.e. $\operatorname{supp} f = \{\boldsymbol{x} : f(\boldsymbol{x}) \neq 0 \wedge \boldsymbol{x} \in \Omega^0\}$.

As shown in Figure 3.4, at each level $l = 1 \ldots M$ the boundary of Ω^l may be aligned with the meshlines of V^l or V^{l-1}. These situations are called *weak condition* and *strong condition* respectively. While it is possible to work assuming just the satisfaction of the weak condition, it may lead to some cases where all the weights associated to finer basis functions are zero. Another point that we need to consider is that we need to replace

the coarse basis functions with finer ones, so if the area to be refined is smaller than the support of a coarse basis function, the weights associated to the finer basis functions will again be zero. Due to this, the areas we select for refinement must satisfy some additional constraints. To avoid these kind of problems and have a correspondence between the mesh and the basis we give the following definition [28, p. 3560] :

Definition 6. In the Hierarchical setting, we will call a mesh *admissible* if at all levels the area selected for refinement satisfies the strong condition on domain boundaries and Ω^l is defined as the union of the supports of previous-level basis functions $N \in \mathcal{N}^{l-1}$.



(a) Strong condition on domain boundaries. The boundary of Ω^{l+1} is aligned with the knot lines of $\mathcal{V}^l$.

(b) Weak condition on domain boundaries. The boundary of Ω^{l+1} is aligned with the knot lines of $\mathcal{V}^{l+1}$.

Figure 3.4: Hierarchical setting: Different conditions on the domain boundaries.

A smart choice when it comes to substituting coarse basis functions with finer ones is exploiting the subdivision property of B-splines. A univariate B-spline N_Ξ defined on the knot vector $\Xi = [\xi_1, \xi_2 \dots \xi_{p+2}]$ can be expressed as a linear combination of scaled copies of itself through the following formula [23, 30] :

$$N_\Xi(\xi) = \sum_{i=1}^{p+2} 2^{-p} \binom{p+1}{i-1} N_\Xi(2\xi - \xi_i) \tag{3.5}$$

Note that $N_\Xi(2\xi - \xi_i) = N_{\Xi_i}$ where the new knot vector Ξ_i is constructed from Ξ halving al the knot spans and taking $p+2$ subsequent knots. For example, for a quadratic basis function defined on $\Xi = [0, 1, 2, 3]$ we would have

$$\Xi_1 = [0, 0.5, 1, 1.5] \quad \Xi_2 = [0.5, 1, 1.5, 2]$$
$$\Xi_3 = [1, 1.5, 2, 2.5] \quad \Xi_4 = [1.5, 2, 2.5, 3]$$

The relation given in Equation (3.5) is at the core of the Hierarchical refinement: It tells us which functions of level $l + 1$ we need to include in the basis when removing functions from level l, thus ensuring the nestedness of the Hierarchical spaces, and, if one wants to utilize the weighted basis, also explicitly gives the correct coefficients needed to maintain the partition of unity. Note that we will not use the weighted basis in the numerical examples of Section 4; this is to be consistent with the definitions given in the papers we used as references, [11, 15, 28].

Once an existing function is subdivided using Equation (3.5), we need to check if any of the components is already present in the list of basis functions. If a component is not present, we add it to the list; if it is then we move to the next component and repeat the check. Note that if we want to utilize the weighted basis and a component is present in the list of functions, we would need to update its weight by adding the new coefficient.

The extension to the bivariate case is straightforward: Each bivariate basis function is a tensor product of two univariate ones. We then apply Equation (3.5) to each of them and then construct all the resulting bivariate components via the usual tensor product.

Given the conditions for the selection of areas to refine stated above, Equation (3.5) allows for a more intuitive understanding of how the Hierarchical refinement works: When we refine a certain region of the mesh of level l, instead of considering the full tensor product basis of level $l+1$ and selecting the correct functions from there, we can now see it as just substituting the old level l functions with their corresponding components in level $l+1$. This point of view is interesting because it shifts the attention from refining *elements* to refining *functions*.

We are now ready to construct the Hierarchical basis:

Definition 7. The *Hierarchical B-spline basis* $\mathcal{H}$ is recursively constructed as follows:

1. Initialization: $\mathcal{H}^0 = \{N \in \mathcal{N}^0 : \operatorname{supp} N \neq \emptyset\}$
2. Recursive case: $\mathcal{H}^{l+1} = \mathcal{H}_A^{l+1} \cup \mathcal{H}_B^{l+1}$ for $l = 0, \dots, M-1$, where
$$\mathcal{H}_A^{l+1} = \{N : N \in \mathcal{H}^l, \operatorname{supp} N \nsubseteq \Omega^{l+1}\}$$
$$\mathcal{H}_B^{l+1} = \{N : N \in \mathcal{N}^{l+1}, \operatorname{supp} N \subseteq \Omega^{l+1}\}$$
3. $\mathcal{H} = \mathcal{H}^M$

Note that the above definition does not include the weights. The recursive definition ensures that we always select the correct functions to include in the basis. The first step initializes the Hierarchical basis with all the relevant functions of the underlying tensor product basis $\mathcal{N}^0$. The recursive procedure then updates the basis by removing the coarse functions contained inside the refined region and including the finer ones substituting them.

Figure 3.5 presents some of the basis functions defined on the same mesh used in Figure 3.3. In the Classical Hierarchical case, all the functions have rectangular support since they are plain tensor product of univariate functions.

As a result of the definition, the Classical Hierarchical B-spline basis and the associated spaces have the following properties, as proved in [28]:

- The functions in $\mathcal{H}$ are linearly independent.
- The spaces spanned by the basis are nested, i.e. $\operatorname{span}\mathcal{H}^l \subseteq \operatorname{span}\mathcal{H}^{l+1}$.

We stress that without weighting the functions the Classical Hierarchical basis does not constitute a partition of unity.

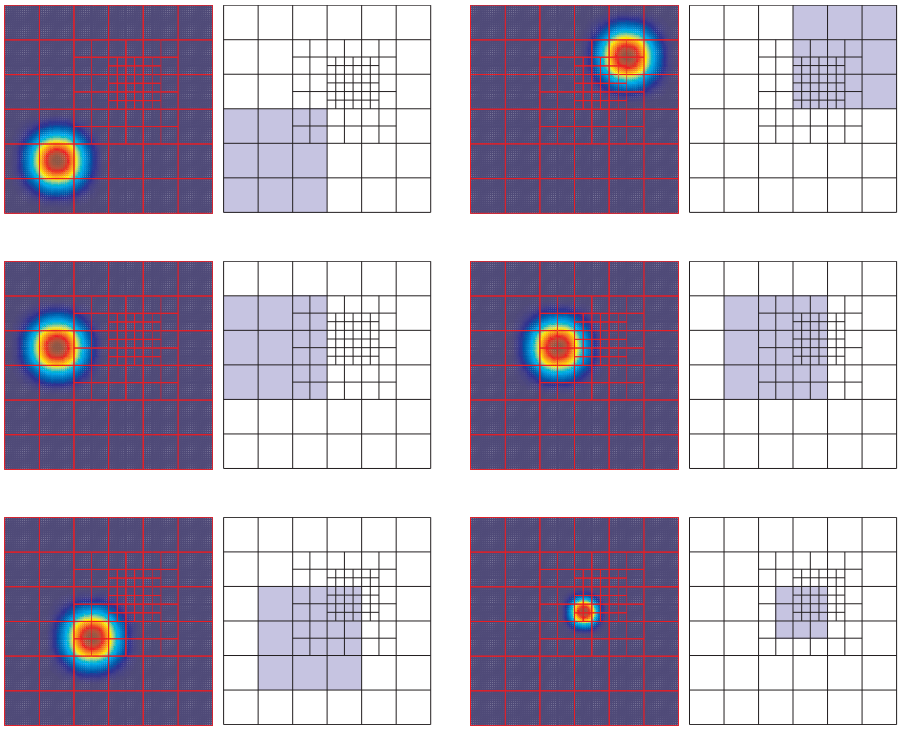


Figure 3.5: Classical Hierarchical Basis: Some of the biquadratic basis functions defined on the same mesh used in Figure 3.3. For each function, on the left is presented a top view of the evaluation plot and on the right the elements constituting its actual support.

3.2.3 The Truncated Hierarchical Basis

While the Hierarchical B-splines presented above provide good flexibility and allow for localized refinement, the number of overlapping basis functions can increase very rapidly with the introduction of new levels. This happens because the large support of the coarse basis functions may overlap with the support of several fine-scale ones. See for example in Figure 3.5, where the top-right function, defined at level 0, overlaps with all the fine-scale functions in level 2.

This behaviour has a negative impact on the formation and solution of linear system of algebraic equation associated to the solution of the discrete (finite element) variational problem: A higher number of overlaps means we need to perform more functions evaluations and add more elements in the system matrices. This on one side increases the assembly time required to build such matrices, and on the other side it affects the sparsity, spectrum, and conditioning number of the matrices, with consequences on the performance of iterative solvers. In order to address these problems, a new basis for the Hierarchical space was proposed in [11]. The key idea is that we can appropriately *truncate* the coarse basis functions, thus reducing their support and significantly decreasing the number of overlaps.

The truncation of a basis function is defined as follows:

Definition 8. Let T be a basis function defined at level l, and let

$$T = \sum_{j \,:\, N_j \in \mathcal{N}^{l+1}} \alpha_j \, N_j$$

be its representation respect to the fine-scale basis associated to level $l+1$. The *truncation*

of T respect to $\mathcal{N}^{l+1}$ and Ω^{l+1} is defined as

$$\operatorname{trunc}^{l+1} T = \sum_{\substack{j\,:\,N_j \in \mathcal{N}^{l+1}, \\ \operatorname{supp} N_j \not\subseteq \Omega^{l+1}}} \alpha_j \, N_j \tag{3.6}$$

It is clear that the coefficients α_j depend not only on the component N_j they refer to, but also on the function T considered. We omitted the explicit dependance to ease the notation.

The Truncated Hierarchical basis is then defined as follows [11] :

Definition 9. The *Truncated Hierarchical B-spline basis* $\mathcal{T}$ is recursively constructed as follows:

1. Initialization: $\mathcal{T}^0 = \mathcal{H}^0$
2. Recursive case: $\mathcal{T}^{l+1} = \mathcal{T}_A^{l+1} \cup \mathcal{T}_B^{l+1}$ for $l = 0, \ldots, M-1$, where
$$\mathcal{T}_A^{l+1} = \{\operatorname{trunc}^{l+1} T : T \in \mathcal{T}^l \wedge \operatorname{supp} N \nsubseteq \Omega^{l+1}\}$$
$$\mathcal{T}_B^{l+1} = \mathcal{H}_B^{l+1}$$
3. $\mathcal{T} = \mathcal{T}^M$

Note that the representation of T in terms of next-level functions is easily obtained through Equation (3.5). The truncation mechanism removes all those components of T that are explicitly included in the basis by the recursive step of the definition. This procedure appropriately shrinks the support of all functions that cross over multiple levels in the mesh, effectively reducing the number of overlaps. Also note that the way of expressing the truncation as given in Equation (3.6) is what we will call an *additive* representation. It is also possible to use a *subtractive* representation, expressing the truncation as

$$\operatorname{trunc}^{l+1} T = T - \sum_{\substack{j\,:\,N_j \in \mathcal{N}^{l+1}, \\ \operatorname{supp} N_j \subseteq \Omega^{l+1}}} \alpha_j \, N_j \tag{3.7}$$

Both these representations have advantages and disadvantages which are discussed in Section 4.

Figure 3.6 shows the Truncated Hierarchical basis constructed on the same mesh as in Figure 3.1. Note that no weights are needed to maintain the partition of unity.

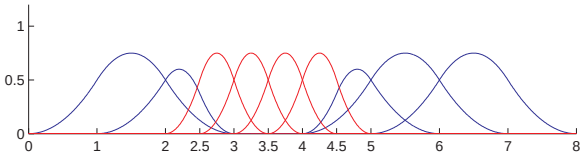


Figure 3.6: Truncated Hierarchical Basis: The quadratic basis on the same mesh as in Figure 3.1. Partition of unity is automatically achieved by the truncation procedure.

Figure 3.7 presents the same basis functions as in Figure 3.5 with the truncation procedure applied. As we can see, the support of each Truncated function is modified in order to reduce the number of overlaps with finer levels. This, however, makes some functions lose the rectangular shape of their support.

The Truncated Hierarchical basis naturally inherits the properties of the Classical Hierarchical basis, and also adds some more. In particular:

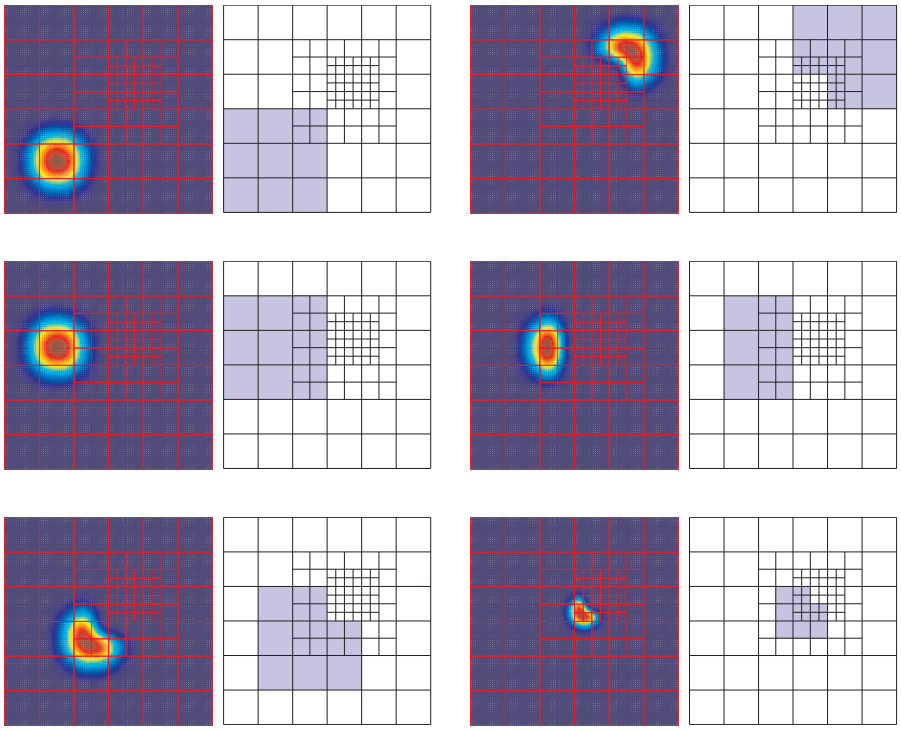


Figure 3.7: 2D Truncated Hierarchical basis: The biquadratic basis constructed on the same mesh, and the corresponding functions, as in Figure 3.5. For each function, on the left is presented a top view of the evaluation plot and on the right the elements constituting its actual support.

- The functions in $\mathcal{T}$ are linearly independent.
- The spaces are nested, i.e. $\text{span}\mathcal{T}^l \subseteq \text{span}\mathcal{T}^{l+1}$.
- The basis maintains partition of unity (without needs for weights).

In addition, if we consider the Classical Hierarchical basis $\mathcal{H}$ defined on the same mesh as $\mathcal{T}$, then:

- The cardinality of the basis is the same: $|\mathcal{H}| = |\mathcal{T}|$.
- The spaces spanned are the same: $\text{span}\mathcal{H} = \text{span}\mathcal{T}$.

Proofs for the above can be found in [11].

3.3 LR B-splines

LR B-splines were recently proposed by Dokken et al. in [6] and later applied to Isogeometric Analysis by Johannessen et al. in [15]. We report here some of the theory contained in those papers, while taking a different approach that focuses on clarity and ease of understanding.

LR B-splines differentiate themselves from the Hierarchical cases in the way the refinement is applied: while Hierarchical functions rely on the subdivision rule given in Equation (3.5) and generate up to $p + 2$ new functions from each original B-spline, LR

B-splines use the knot insertion procedure, inserting one knot at a time and splitting old B-splines into 2 new ones. The fact the the knots are inserted one at a time is crucial, especially in the bivariate setting: We will show that even when inserting the same knot values as produced by the subdivision rule, the resulting refined B-spline basis may be different.

LR B-splines are locally refined in the same way the standard tensor-product B-splines are. From basic spline theory we know that it is possible to perform knot insertion to enrich the spline space while leaving the geometry description unchanged. In the univariate case, if we want to insert the knot $\hat{\xi}$ between the knots ξ_{i-1} and ξ_i we have

$$B_\Xi(\xi) = \alpha_1 B_{\Xi_1}(\xi) + \alpha_2 B_{\Xi_2}(\xi) \tag{3.8}$$

where

$$\begin{aligned} \alpha_1 &= \begin{cases} \frac{\hat{\xi}-\xi_1}{\xi_{p+1}-\xi_1} & \xi_1 \leqslant \hat{\xi} \leqslant \xi_{p+1} \\ 1 & \xi_{p+1} \leqslant \hat{\xi} \leqslant \xi_{p+2} \end{cases} \\ \alpha_2 &= \begin{cases} 1 & \xi_1 \leqslant \hat{\xi} \leqslant \xi_2 \\ \frac{\xi_{p+2}-\hat{\xi}}{\xi_{p+2}-\xi_2} & \xi_2 \leqslant \hat{\xi} \leqslant \xi_{p+2} \end{cases} \end{aligned} \tag{3.9}$$

and the knot vectors are

$$\begin{aligned} \Xi &= [\xi_1, \xi_2 \dots \xi_{i-1}, \quad \xi_i \dots \xi_{p+1}, \ \xi_{p+2}] \\ \Xi_1 &= [\xi_1, \xi_2 \dots \xi_{i-1}, \hat{\xi}, \ \xi_i \dots \xi_{p+1} \quad\ ] \\ \Xi_2 &= [\quad \xi_2 \dots \xi_{i-1}, \hat{\xi}, \ \xi_i \dots \xi_{p+1}, \ \xi_{p+2}] \end{aligned}$$

As we can see, inserting one knot splits the original B-spline into two new B-splines described by the local knot vectors Ξ_1 and Ξ_2. The weights α_1 and α_2 are needed to maintain partition of unity. Figure 3.8 shows some examples of the application of Equation (3.8).

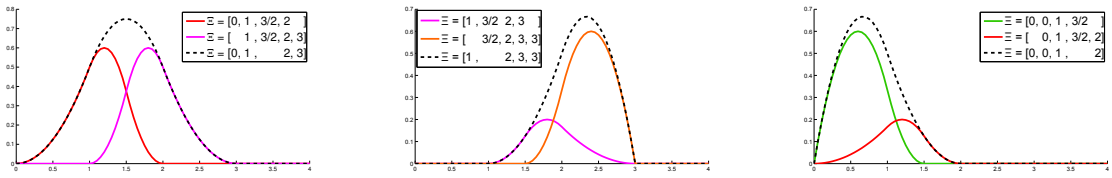


Figure 3.8: LR B-splines: Examples of knot insertion for $\hat{\xi} = \frac{3}{2}$. Dashed lines: The original functions. Colors: The new functions resulting from the splitting.

In the bivariate case, functions are refined one parametric direction at a time. In this case we obtain:

$$\begin{aligned} B_{\boldsymbol{\Xi}}(\xi, \eta) &= B_\Xi(\xi)\, B_{\mathcal{H}}(\eta) \\ &= (\alpha_1 B_{\Xi_1}(\xi) + \alpha_2 B_{\Xi_2}(\xi))\ B_{\mathcal{H}}(\eta) \\ &= \alpha_1 B_{\boldsymbol{\Xi_1}}(\xi, \eta) + \alpha_2 B_{\boldsymbol{\Xi_2}}(\xi, \eta) \end{aligned} \tag{3.10}$$

In the following we will call *meshline extension* all mesh-altering actions like inserting a new meshline, prolonging existing meshlines (possibly connecting two existing ones) or increasing the multiplicity of meshlines. When a new meshline extension is inserted, we need to know which basis functions are affected by it. For this purpose, we give the following definition:

Definition 10. A meshline ϵ is said to *traverse the support* of a function $B_{[\xi_1 \dots \xi_{p_1+2};\eta_1 \dots \eta_{p_2+2}]}$ if

- ϵ is a horizontal line $\epsilon = [\xi_1^*, \xi_2^*] \times \eta^*$ such that

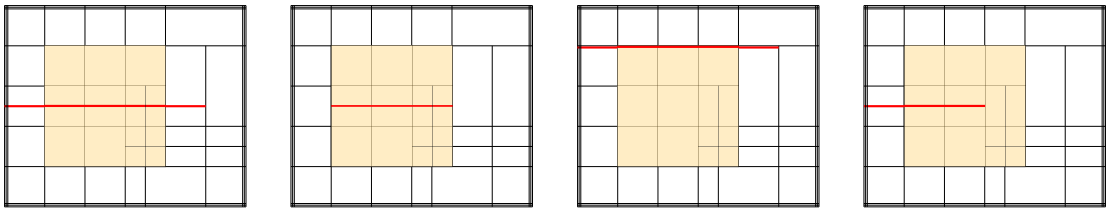
$$\xi_1^* \leqslant \xi_1, \quad \xi_{p_1+2} \leqslant \xi_2^*, \quad \eta_1 \leqslant \eta^* \leqslant \eta_{p_2+2}$$

- ϵ is a vertical line $\epsilon = \xi^* \times [\eta_1^*, \eta_2^*]$ such that

$$\xi_1 \leqslant \xi^* \leqslant \xi_{p_1+2}, \quad \eta_1^* \leqslant \eta_1, \quad \eta_{p_2+2} \leqslant \eta_2^*$$

In particular, a horizontal line is said to traverse *the interior* of $B_{[\xi_1 \dots \xi_{p_1+2};\eta_1 \dots \eta_{p_2+2}]}$ if $\eta_1 < \eta^* < \eta_{p_2+2}$ and traverse *the edge* if $\eta^* = \eta_1$ or $\eta^* = \eta_{p_2+2}$. Similarly, a vertical line is said to traverse the interior if $\xi_1 < \xi^* < \xi_{p_1+2}$ and traverse the edge if $\xi^* = \xi_1$ or $\xi^* = \xi_{p_1+2}$.

Figure 3.9 shows some examples of lines traversing the support of a basis function.



(a) Line traversing the interior of B

(b) Line traversing the interior of B

(c) Line traversing the edge of B

(d) Line neither traversing the edge nor the interior of B

Figure 3.9: LR B-splines: Examples of lines traversing the support of a basis function.

When a meshline extension is applied, the refinement process is composed of two steps:

1. Split any function which support is traversed by the *new* meshline.
2. For all new functions, check if their support is traversed by any *existing* meshline, and split again if this happens.

In step 1 we test all current functions against one meshline. In step 2 we test all newly created functions against all existing meshlines. Note that when the meshline extension is an actual elongation, possibly connecting two separate existing meshlines, we will use the full length of the resulting line to test the functions for splitting. When a function is flagged for splitting, this is performed through the use of Equations (3.9) and (3.10).

In view of the above, we can give the following two definitions:

Definition 11. In the LR B-splines setting, an *admissible mesh* is any mesh which can be obtained by a sequence of meshline extensions starting from an initial tensor product mesh. Each extension must cause at least one basis function to be split, and the meshlines must end at existing knot values (they cannot stop at the centre of an element). All tensor product meshes are admissible.

Definition 12. An *LR B-spline* is a function which results from the application of the refinement scheme and Equations (3.8)-(3.9). All tensor product B-splines are LR B-splines.

Figure 3.10 shows the 1D LR B-splines basis defined on the same mesh as in the previous examples at Figures 3.1, 3.2, and 3.6. Note that in the univariate setting the LR B-spline refinement coincide with the normal knot insertion. In this case all the weights sum up to 1 and so the LR B-splie basis is the normal B-splines basis originating from the non-uniform knot vector $\Xi = [0, 1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8]$, and automatically maintains partition of unity. Also note that in the general LR B-spline setting the notion of levels is not as present as in the Hierarchical setting; we can however define the level of an LR B-spline function using the maximum knotspan contained in its local knot vector.

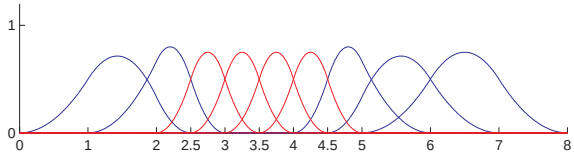


Figure 3.10: LR B-plines: The basis constructed on the same mesh as in figures 3.1, 3.2, and 3.6.

For a thorough example in the bivariate case we refer the reader to [15, p. 481-483].

Given an initial tensor product mesh $\mathcal{M}_0$, a sequence of meshline extensions $\{\epsilon_i\}_{i=1}^n$ and corresponding admissible meshes $\mathcal{M}_i = \{\mathcal{M}_{i-1} \cup \epsilon_i\}$ and LR B-splines $\mathcal{L}^i$, the following properties hold [6, 15]:

- The spaces are nested: $\text{span}\mathcal{L}^i \subseteq \text{span}\mathcal{L}^{i+1}$.
- The LR B-splines defined on a mesh are not affected by the order in which the meshline extensions have been inserted, i.e. if $\mathcal{M}$ and $\hat{\mathcal{M}}$ are two identical LR B-splines meshes that differ only for the order in which the meshlines extensions have been applied, then the resulting LR B-splines functions are the same.
- The LR B-splines form a partition of unity.

Note that, as of today, no formal proof for the linear independence of the LR B-splines functions has been given. This is mostly due to the fact that the single-line insertion mechanism used in this setting allows for several different types of refinement strategies, and the particular choice naturally affects the spline space. Indeed, in some cases it may happen that the resulting set of LR B-splines is linearly dependant; however, there are also several ways to recover the linear independency as proposed in [6, 15]. The linear independence of LR B-splines depending on the type of refinement strategy used is currently object of research.

In all our examples we will use the *Structured Mesh* refinement presented in [15]. This strategy focuses on refining functions, instead of elements. The idea of refining elements is indeed a legacy from the Finite Element methodology. Using the Structured Mesh approach, one instead selects which basis functions to refine. This can be done through the use of custom-built criteria, just as one would do in an adaptive refinement scheme. The idea proposed in [15] is to compute the error pertaining to each basis function as

$$\|e\|^2_{\text{supp}B_i} = \sum_{K \in \text{supp}B_i} \|e\|^2_K \tag{3.11}$$

i.e. we define the *B-spline error* as the sum of the normal error $\|e\|$ measured in the energy norm over all elements in the support of B_i. Once the functions to be refined are identified, we proceed to insert several knot lines in both directions, halving the knot spans

of the largest supported knot interval. Note that the Structured Mesh strategy will yield the same results on the mesh as the subdivision rule used in the Hierarchical setting. This means that all meshes which are admissible in the Hierarchical setting, i.e. they satisfy the conditions of Definition 6, are also admissible in the LR B-splines setting and can be obtained using the Structured Mesh refinement. For this reason we have always used this approach for our examples, as it provides a better ground for comparison.

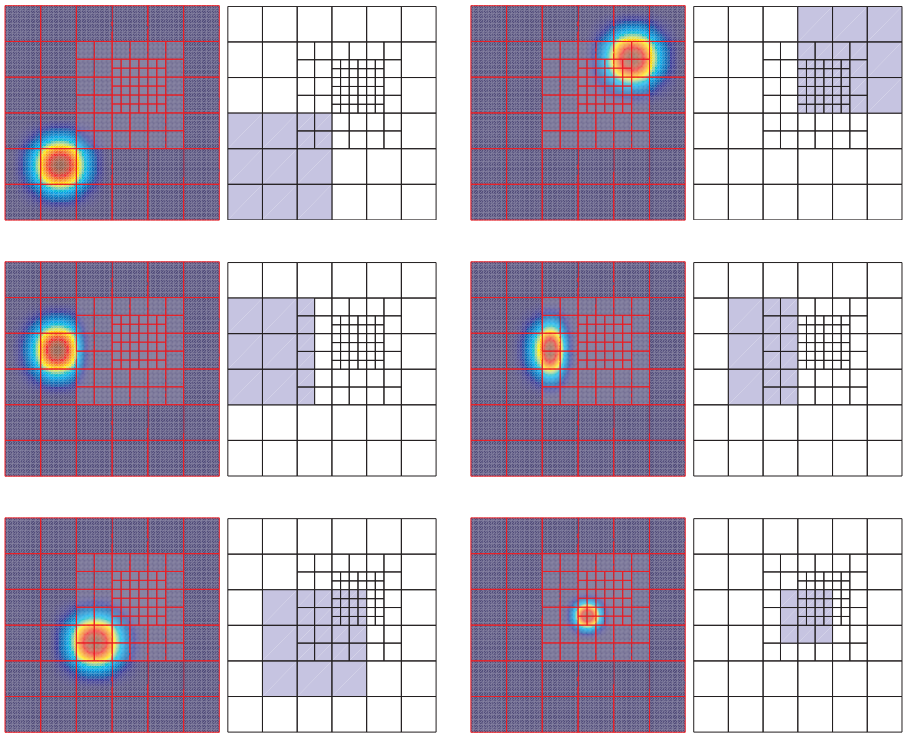


Figure 3.11: LR B-splines: The biquadratic basis constructed on the same mesh, and the corresponding functions, as in Figure 3.5 and 3.7. For each function, on the left is presented a top view of the evaluation plot and on the right the elements constituting its actual support.

4

Results

We present here the results of our analysis on the different type of basis functions outlined in Section 3. The *Qualitative analysis* sections collects results regarding the mathematical properties of the various basis, many of which were already briefly listed in the corresponding sections. The *Quantitative analysis* section focuses on implementation and numerical quantities and discusses the properties of the stiffness and mass matrices generated using the different splines functions.

4.1 Qualitative analysis

We would like to start pointing out that, under normal mesh refinement iterations (i.e. excluding special constructed cases), on the qualitative level all the three classes of splines give comparable results. However, there are some interesting distinctive features that are worth to be mentioned.

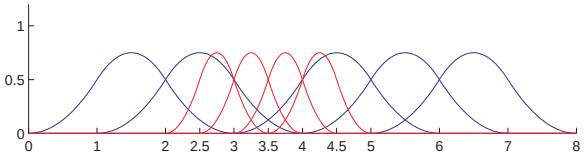
4.1.1 Different functions

The functions included in the Classical Hierarchical, Truncated Hierarchical, and LR B-splines sets are fundamentally different: Classical Hierarchical functions (Definition 7) are standard uniform tensor product B-splines; Truncated Hierarchical functions (Definitions 8 and 9) are generally a linear combination of uniform tensor product B-splines; LR B-splines functions (Equation (3.8) and Definition 12) are non-uniform tensor product B-splines. With the notation introduced at page 31 we have that the general functions $H \in \mathcal{H}$, $T \in \mathcal{T}$, and $L \in \mathcal{L}$ can be written as

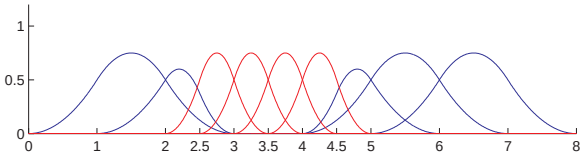
$$H = N$$
$$T = \sum \alpha_i N_i$$
$$L = \alpha B$$

for appropriate indices i and weights α.

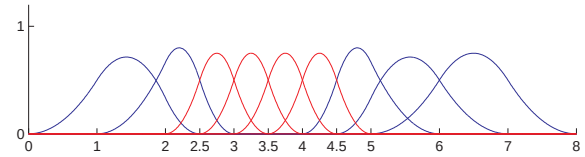
Figure 4.1 shows the Classical Hierarchical, Truncated Hierarchical and LR B-splines basis for the knot vector $\Xi = [0, 1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8]$. Note that on the uniform vector $[0 : 8]$ all three families of functions would be exactly the same. In the Classical Hierarchical case, partition of unity is not preserved. In the Truncated Hierarchical case



(a) Classical Hierarchical



(b) Truncated Hierarchical



(c) LR B-splines

Figure 4.1: Qualitative analysis: The different quadratic bases constructed using the same knot vector.

this is achieved automatically by the truncation procedure, which removes some components from the old-level functions. The LR B-splines are instead directly defined on non-uniform knot vectors, and also use weights to maintain partition of unity.

Figure 4.2 shows a comparison of the support for some of the basis functions presented in the Figures 3.5, 3.7, and 3.11.

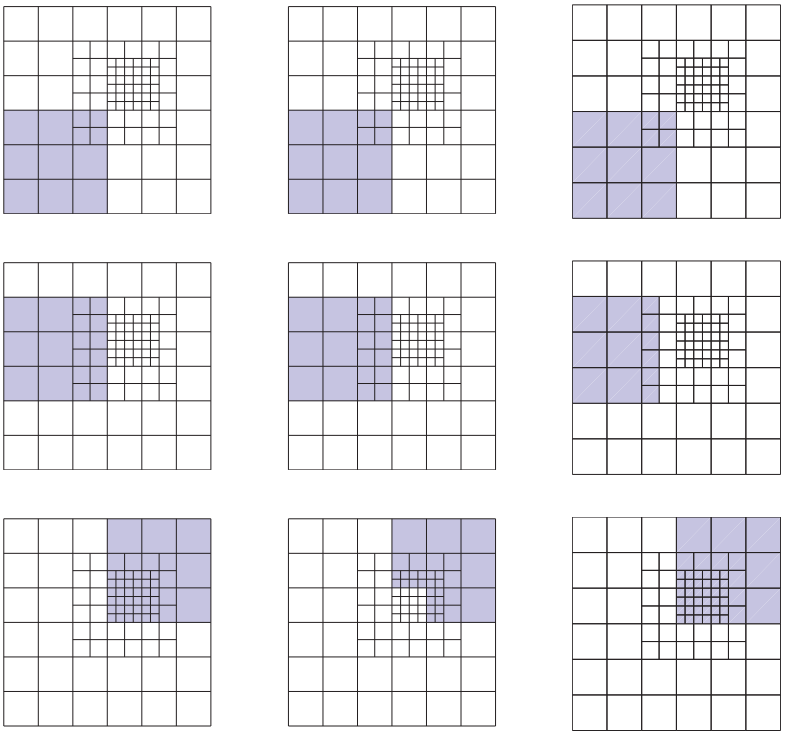
4.1.2 Different spaces

Perhaps the most important and interesting difference is that for some meshes the Hierarchical basis and the LR B-splines set span different spaces. One such example is given in figure 4.3.

The central function appearing in the Classical and Truncated Hierarchical setting corresponds to two distinct functions in the LR B-splines set. This is due to the way the refinement works in the LR B-splines setting: As we can see there is one new meshline which completely traverses the support of the central function. When this meshline is inserted, it triggers the LR B-splines refinement algorithm which splits the original B-spline into two new ones as expected. This does not happen in the Hierarchical framework, which leaves the function unchanged in the classic case or appropriately reduces its support in the Truncated case.

Mourrain [20] presented a formula for the maximum dimension of the space of piecewise polynomials with given continuity on a mesh: given a planar mesh with F faces (the elements), H horizontal and V vertical internal edges and P vertices, the maximum dimension of the space of bivariate piecewise polynomials of degrees (p, q) with continuity (k, l) along element edges is given by

$$\mathcal{S} = (p+1)(q+1)F - (p+1)(l+1)H - (q+1)(k+1)V + (k+1)(l+1)P + h \quad (4.1)$$



(a) Left Column: Classical Hierarchical

(b) Centre Column: Truncated Hierarchical

(c) Right Column: LR B-splines

Figure 4.2: Qualitative analysis - Different Functions: The support of corresponding biquadratic basis functions in the three spline families presented in the Figures 3.5, 3.7, and 3.11.

where h is the homology factor of the mesh, which is equal to zero for all the refinement schemes used here. To the best knowledge of the authors there is no formal proof that the Classical Hierarchical, Truncated Hierarchical or LR B-splines actually span the entire space.

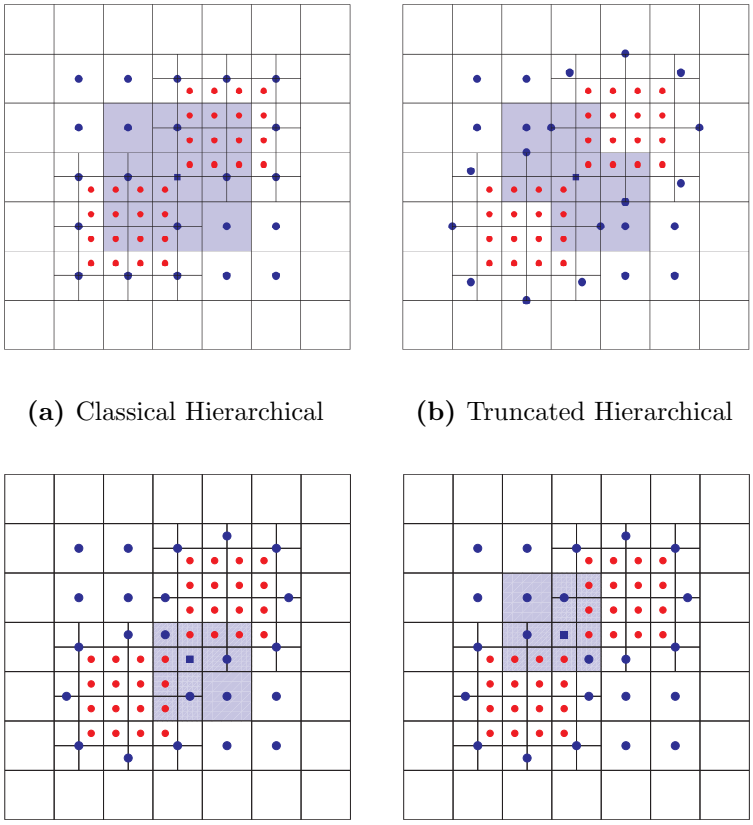
4.1.3 Different refinement strategies

Hierarchical functions rely on Equation (3.5) to apply the refinement. This procedure halves the local knot vectors of the function and replaces the original B-spline with up to $p + 2$ new functions in the univariate case, or $(p + 2)(q + 2)$ in the bivariate case.

LR B-splines use the knot insertion given in Equation (3.8), which introduces two new B-splines functions. This procedure allows for more flexibility in the refinement approach as there are no prescriptions on the number or positions of the new knots. In addition to the Structured Mesh strategy, already presented in Section 3, in [15] two other different refinement strategies are proposed: *Minimum Span* and *Full Span*. While all these strategies insert the meshlines so that they halve the knotspans, this is not a requirement as the use of non-uniform knot vectors is already built-in in the definitions of LR B-splines.

The Minimum Span strategy aim is to keep the refinement as localized as possible. Once an element is marked for refinement, a cross is inserted through its centre and the meshlines are made to be as short as possible, while still splitting at least one function.

In the Full Span strategy the idea is to split *all* B-splines with support on a selected



(a) Classical Hierarchical **(b)** Truncated Hierarchical

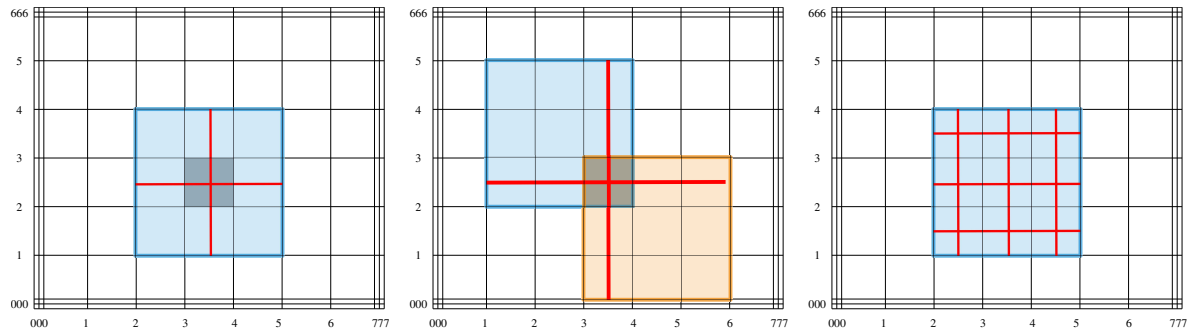
(c) LR B-splines, first function **(d)** LR B-splines, second function

Figure 4.3: Qualitative analysis - Different spaces: An example of mesh on which the biquadratic Hierarchical bases span different spaces than the LR B-splines basis. The central level 0 function in the Hierarchical cases corresponds to two distinct functions in the LR B-splines basis. Both the Hierarchical bases are constituted of 55 functions; the LR B-splines basis contains 56 functions. The highlighted area is the support of the selected function, represented with a square as anchor symbol. The anchors are placed as described at page 33.

element. This is done inserting two meshlines in a cross through the centre of the element. The new meshlines will have to span from the minimum to the maximum knot values of all functions with support on the marked element in both parametric direction. This strategy makes sure that all B-splines with support on the marked element are treated equally, but on the other hand this results in an extension of the refinement away from the selected element, in particular for high polynomial degrees.

A common drawback of both the Full and Minimum Span is that some elements will be traversed by only one meshline and therefore will be split into two rectangular elements, effectively doubling their aspect ratio.

While the possibility of applying other refinement strategies is allowed by the definitions and the theory of LR B-splines, some may lead to linearly dependant sets. The research in these cases is still ongoing and, as of today, the Structured Mesh is the best candidate for a stable refinement algorithm.



(a) Minimum Span **(b)** Full Span (only two functions are shown) **(c)** Structured Mesh

Figure 4.4: Qualitative analysis: Different types of refinement strategies using LR B-splines.

4.1.4 Different admissible meshes

A direct consequence of the various refinement approaches available with LR B-splines is that some meshes which are legal in a LR setting cannot be reproduced using Hierarchical splines. On the other hand, LR B-splines are not capable of achieving some configurations of the weak conditions on domain boundaries available in the Hierarchical framework. For an example see Figure 3.4b. In that case, the meshlines of Ω^1 are stopping in the centre of the elements, a behaviour that is disallowed by the LR definitions.

Another difference is that LR B-splines are currently defined starting from a global tensor-product mesh, i.e. only on rectangular parametric domains. Conversely, Hierarchical B-splines can be defined on non-rectangular parametric domains.

Meshes that are defined on a rectangular domain and also satisfy the conditions of Definition 6 are admissible in both the Hierarchical and LR B-splines framework.

4.2 Quantitative analysis

Here, we first discuss details related to different representation of Truncated Hierarchical basis, and then present the numerical results obtained for different meshes and polynomial degrees,

4.2.1 Representation of Truncated functions

As we briefly mentioned earlier, Truncated Hierarchical B-splines can be represented in an additive or subtractive fashion; we restate Equations (3.6) and (3.7) for reading convenience:

$$\operatorname{trunc}^{l+1} T = \sum_{\substack{j\,:\,N_j \in \mathcal{N}^{l+1}, \\ \operatorname{supp} N_j \not\subseteq \Omega^{l+1}}} \alpha_j \, N_j \qquad \text{Additive representation} \tag{4.2}$$

$$\operatorname{trunc}^{l+1} T = T - \sum_{\substack{j\,:\,N_j \in \mathcal{N}^{l+1}, \\ \operatorname{supp} N_j \subseteq \Omega^{l+1}}} \alpha_j \, N_j \qquad \text{Subtractive representation} \tag{4.3}$$

where N_j are the components of T with respect to the finer level basis functions and α_j the corresponding weights as given by Equation (3.5).

When implementing the code we found that choosing one representation over the other yield important consequences. In a typical finite element code one has to deal with two important aspects: determining which basis functions are active over a given element, and then evaluating such functions.

To address the former a convenient way to retrieve or store the support of the functions is essential. In the case of B-splines this is generally easy since the support is identified with the local knot vectors. When the B-spline is a standard tensor product, and the support is therefore rectangular, this becomes even easier since one only needs to check the starting and ending points of the local knot vectors. As we have seen, however, Truncated Hierarchical functions do not always have rectangular support, hence we need a representation that allows for an easy way to retrieve it. The subtractive representation (4.3) is unfortunately not very helpful in this sense: the fact that a given component is subtracted does not automatically guarantee that the function itself vanishes in that area. Given an element $E \in \operatorname{supp} T$ we should check if *all* possible components on that element are removed in order to know if $\operatorname{trunc} T$ has support on E or not. The additive representation (4.2), on the other hand, is much more convenient: We can simply loop over all components and check if *any* of them has support on E.

To address the latter point we need an efficient way to evaluate basis functions. This is even more important in an Isogeometric setting: Since the Cox-de Boor algorithm is a typical bottleneck of the code, we would like to perform as few basis evaluations as possible. In this case the additive representation (4.2) is not efficient. In a biquadratic case a representation in terms of next-level basis functions comprises of 16 fine-scale functions. This number clearly increases when increasing the polynomial degree: 25 for bicubic functions, 36 for biquartic etc. In addition, an additive representation may require to store the function in terms of the finest-available scale, which would greatly increase the amount of components needed; let's assume, for simplicity, that this is not the case. To give an example, look at the biquadratic basis functions of Figure 3.7. For each of the Truncated Hierarchical basis functions only 4 components are removed. This means that in an additive representation we would still need to evaluate 12 fine-scale functions in order to compute the value of the B-spline we are interested in. In a subtractive representation we would need to evaluate only 5 functions: The original tensor product B-spline and the 4 components we need to subtract.

To summarize, we have the following:

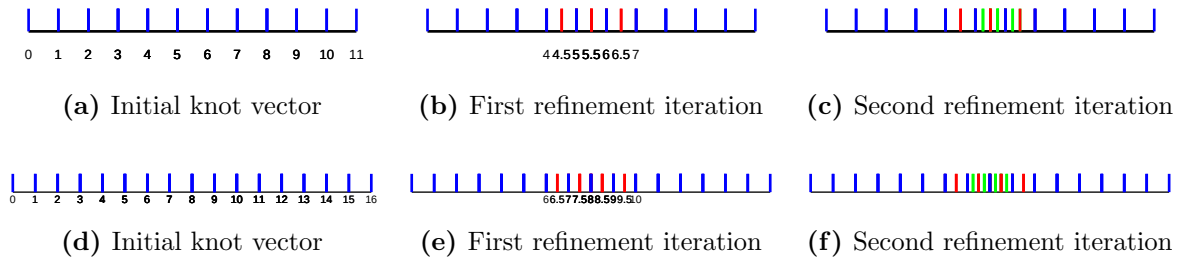
- An additive representation (4.2) is useful when determining the support but not efficient in the function evaluation process;

- A subtractive representation (4.3) does not allow to easily identify the support of the function but is more efficient in its evaluation.

The above discussions are overall considerations: The disadvantages of the representations might be accounted for by programming the algorithm in a smart efficient way. On the other hand, this is still something that needs to be taken into consideration. For an in-depth discussion on the implementation of Truncated Hierarchical B-splines we refer to [17].

4.2.2 1D examples

We now present the results obtained in various 1D examples. We performed several experiments for polynomial degrees $p = 2, 3, 4$, and 5. In each case we started from a uniform, non-open knot vector $\Xi^0 = [0, 1 \dots 5\,p+1]$ and successively applied 6 refinement steps, always refining the central basis function. Note that for odd polynomial degrees a central function always exists, while for even-degree normally there are two functions near the knot vector centre. In this case we chose to always refine the rightmost one. Figure 4.5 shows as examples the first two refinement iterations for $p = 2$ and $p = 3$.



(a) Initial knot vector **(b)** First refinement iteration **(c)** Second refinement iteration

(d) Initial knot vector **(e)** First refinement iteration **(f)** Second refinement iteration

Figure 4.5: 1D Central Refinement: The first three steps of the refinement process in the cases $p = 2$ (above) and $p = 3$ (below). When two functions are equally close to the centre, the rightmost one is selected for refinement.

For each refinement iteration we constructed the stiffness matrix A and the mass matrix M using the Classical Hierarchical, Truncated Hierarchical and LR B-splines functions defined using the same knot vector. We then analysed some important numerical properties of these matrices, namely the sparsity pattern, the conditioning number, and the spectrum. Note that in the univariate case the LR B-splines basis coincides with the standard non-uniform B-splines generated via knot insertion.

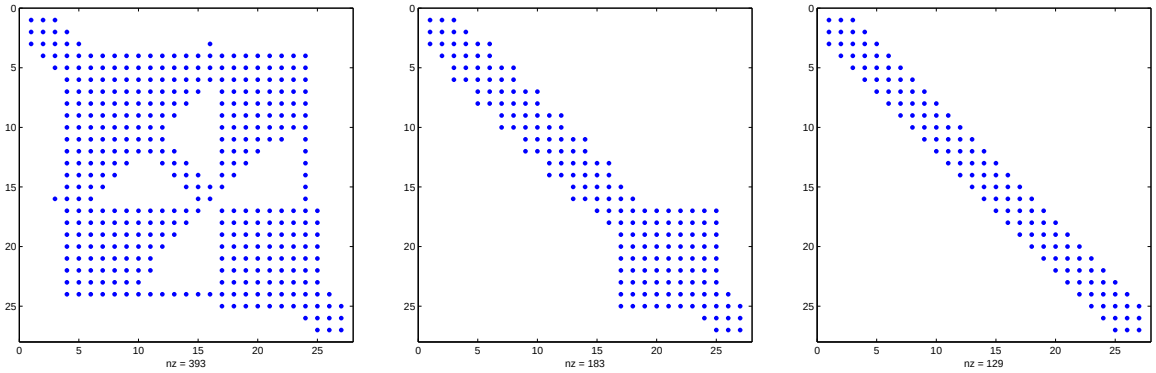
Sparsity Figure 4.6 shows the sparsity patterns of the stiffness matrix at the last refinement iteration after a reordering using the Cuthill-McKee Algorithm [5] has been applied. The top row corresponds to $p = 2$, while to bottom row corresponds to $p = 5$.

As expected the Classical Hierarchical basis functions produce the densest stiffness matrices. This is normal since the support of coarse-level functions remains unaffected by the refinement in neighbouring regions. The values for all polynomial degrees are collected in Table 4.1.

p	Hier.	Trunc.	LR	H/T	H/LR
2	393	183	129	215%	305%
3	803	315	247	255%	325%
4	1257	629	403	200%	312%
5	1919	853	597	225%	321%

Table 4.1: 1D Central Refinement: Number of non-zero elements in the stiffness matrix at the last (6th) refinement iteration. The last two columns present the ratios, rounded to the nearest percentage point.

Conditioning Numbers The conditioning number of the stiffness and mass matrices can be significantly influenced by the way the boundary conditions are imposed. In order



(a) Classical Hierarchical, $p = 2$ **(b)** Truncated Hierarchical, $p = 2$ **(c)** LR B-splines, $p = 2$

(d) Classical Hierarchical, $p = 5$ **(e)** Truncated Hierarchical, $p = 5$ **(f)** LR B-splines, $p = 5$

Figure 4.6: 1D Central Refinement: Examples of sparsity patterns of the stiffness matrices at the last(6th) refinement iteration. The Cuthill-McKee Algorithm has been applied to optimize the bandwidth. Top: $p = 2$. Bottom: $p = 5$.

to avoid any such effect we decided to look at the "pure" conditioning numbers, i.e. before any imposition of the boundary conditions. As is well known, with just pure Neumann boundary conditions the stiffness matrix is singular; the conditioning number can then be defined as the ratio between the largest eigenvalue and the smallest non-zero one. This means that for either the stiffness matrix A or the mass matrix M, given the ordered set of their eigenvalues $[\lambda_1, \lambda_2 \dots \lambda_N]$ we define

$$\begin{aligned} \text{cond}(A) &= \frac{\lambda_N}{\lambda_2} \\ \text{cond}(M) &= \frac{\lambda_N}{\lambda_1} \end{aligned} \tag{4.4}$$

Figure 4.7 shows the plots for the conditioning numbers of both the stiffness and mass matrices for each polynomial degree considered. While all values are quite close to each other, and always remained in the same order of magnitude for our experiments, it is interesting to note that the Truncated Hierarchical and LR B-splines perform very similarly.

Looking at the plots for the stiffness matrix we can see that, with the exception of the lowest degree, i.e. $p = 2$, the conditioning numbers are ordered as

$$\text{cond}(A_T) < \text{cond}(A_{LR}) < \text{cond}(A_H)$$

while for the mass matrix we always have

$$\text{cond}(M_{LR}) < \text{cond}(M_T) < \text{cond}(M_H)$$

where the subscripts indicates the basis functions used.

We can also see that the conditioning numbers of the stiffness matrices are increasing with each refinement iteration, while the conditioning numbers for the mass matrices for $p = 4$ and 5 are bounded from above and below by a constant. This behaviour was already presented by Gahalaut and Tomar in [10], although that result is proven for uniform refinement only.

The Tables 4.2 and 4.3 present the numerical data for $p = 2$ and $p = 3$, respectively.

Stiffness Matrix

Iter.	0	1	2	3	4	5	6
Hier.	12.7425	28.0291	55.7519	111.4035	222.7908	445.5791	891.158
Trunc.	12.7425	25.8255	52.0501	105.3161	213.368	432.4906	876.3622
LR	12.7425	27.2848	55.6005	112.6381	228.1518	462.2306	936.1914

Mass Matrix

Iter.	0	1	2	3	4	5	6
Hier.	46.7947	52.5238	65.8931	116.2265	225.4839	448.1175	894.9733
Trunc.	46.7947	41.5164	42.6706	45.6839	88.2484	176.373	352.7153
LR	46.7947	38.0372	38.4295	38.5944	67.7769	135.5371	271.0706

Table 4.2: 1D Central Refinement: The conditioning numbers for $p = 2$ throughout the mesh refinement. Stiffness Matrix above, Mass Matrix below.

Stiffness Matrix

Iter.	0	1	2	3	4	5	6
Hier.	37.5856	81.2603	162.2944	324.6481	649.3102	1298.6220	2597.2442
Trunc.	37.5856	74.0527	148.1500	296.3336	592.6853	1185.3798	2370.7641
LR	37.5856	75.1932	150.6787	301.4619	602.9764	1205.9794	2411.9722

Mass Matrix

Iter.	0	1	2	3	4	5	6
Hier.	1405.224	1553.052	1585.284	1590.567	1591.561	2238.165	4476.303
Trunc.	1405.224	1292.261	1296.807	1297.363	1297.472	1297.603	2201.907
LR	1405.224	1190.168	1191.548	1191.797	1191.817	1191.819	1191.819

Table 4.3: 1D Central Refinement: The conditioning numbers for $p = 3$ throughout the mesh refinement. Stiffness Matrix above, Mass Matrix below.

Spectrum Figure 4.8 shows the spectra of the stiffness and mass matrices for $p = 2$ at the sixth refinement iteration. The eigenvalues of the stiffness matrix are spread over a large interval, while the eigenvalues of the mass matrix are much more clustered. In all cases the eigenvalues tend to be denser near the origin, but there is no substantial difference between the various basis functions.

Increasing the polynomial degree has different consequences on the eigenvalues of the stiffness and mass matrices: While the large eigenvalues of the stiffness matrix are reduced, those of the mass matrix are increased. The values of the smallest eigenvalues are instead reduced in all cases, as we would expect by the increase in the conditioning numbers. We noted, however, that only a small number of outliers quickly approaches zero, while the other smallest eigenvalues are still reduced but not as fast. In particular, for the Classicalal and Truncated Hierarchical basis functions the smallest eigenvalues seems to decrease faster than those associated with LR B-splines. Figure 4.9 shows the spectra of the stiffness and mass matrices produced with basis functions of degree $p = 5$.

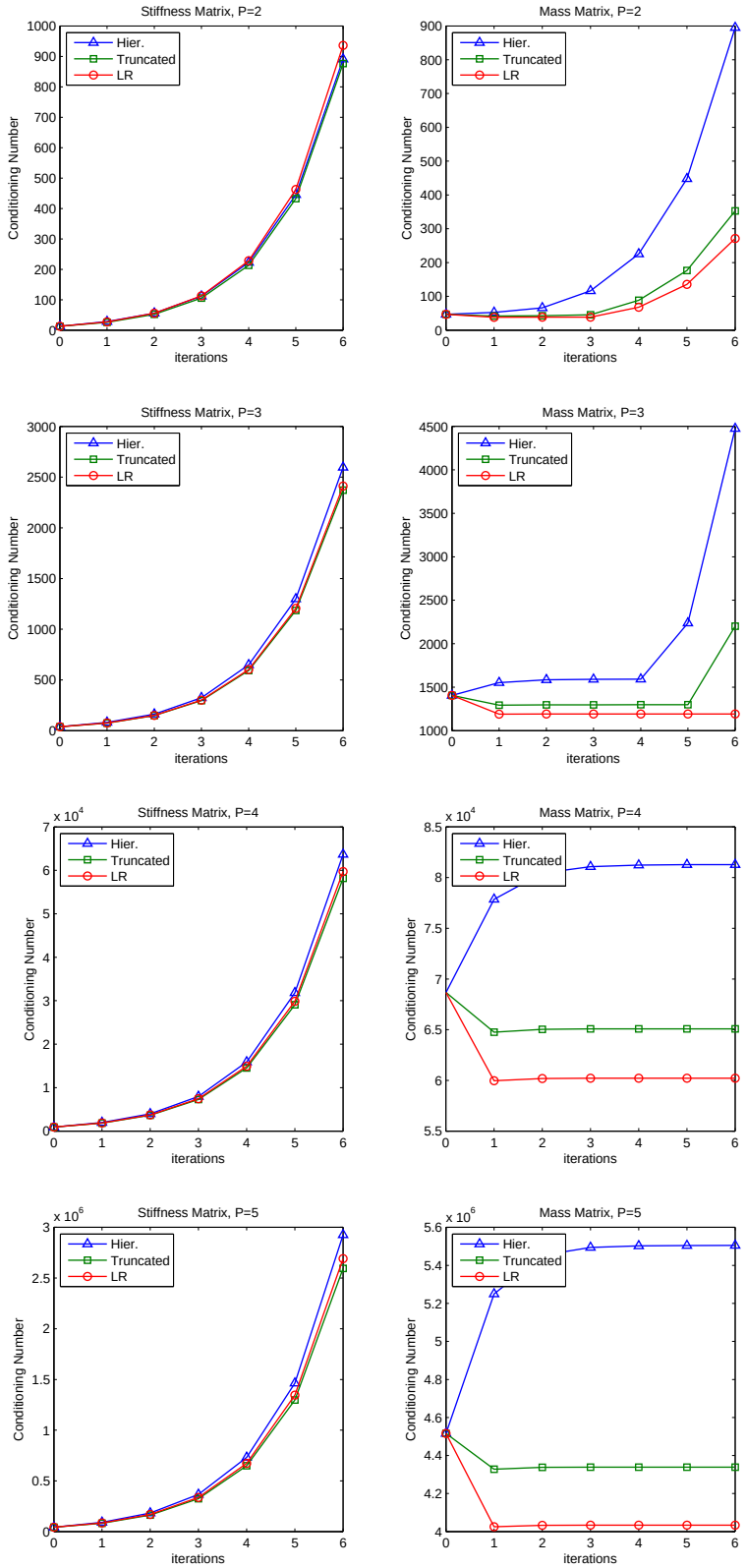


Figure 4.7: 1D Central Refinement: Graphs of the conditioning numbers of stiffness matrices (left column) and mass matrices (right column) from $p = 2$ (top) to $p = 5$ (bottom).

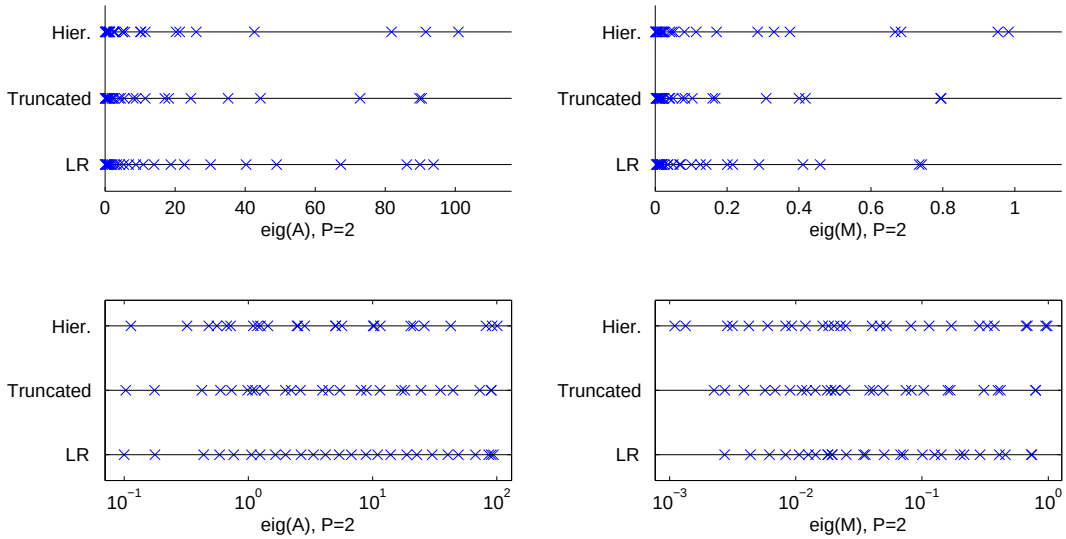


Figure 4.8: 1D Central Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 2$ at the last refinement iteration. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

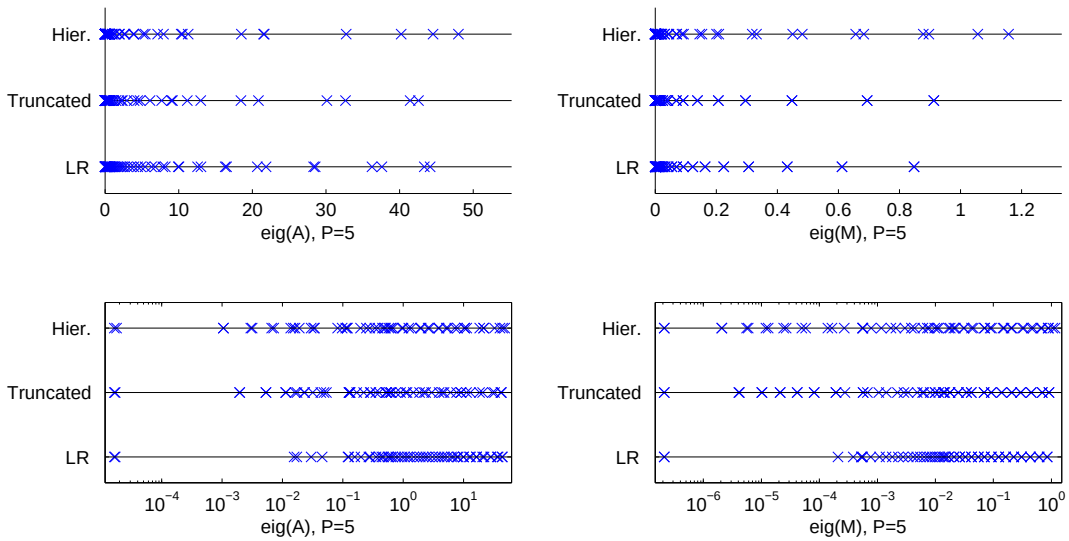
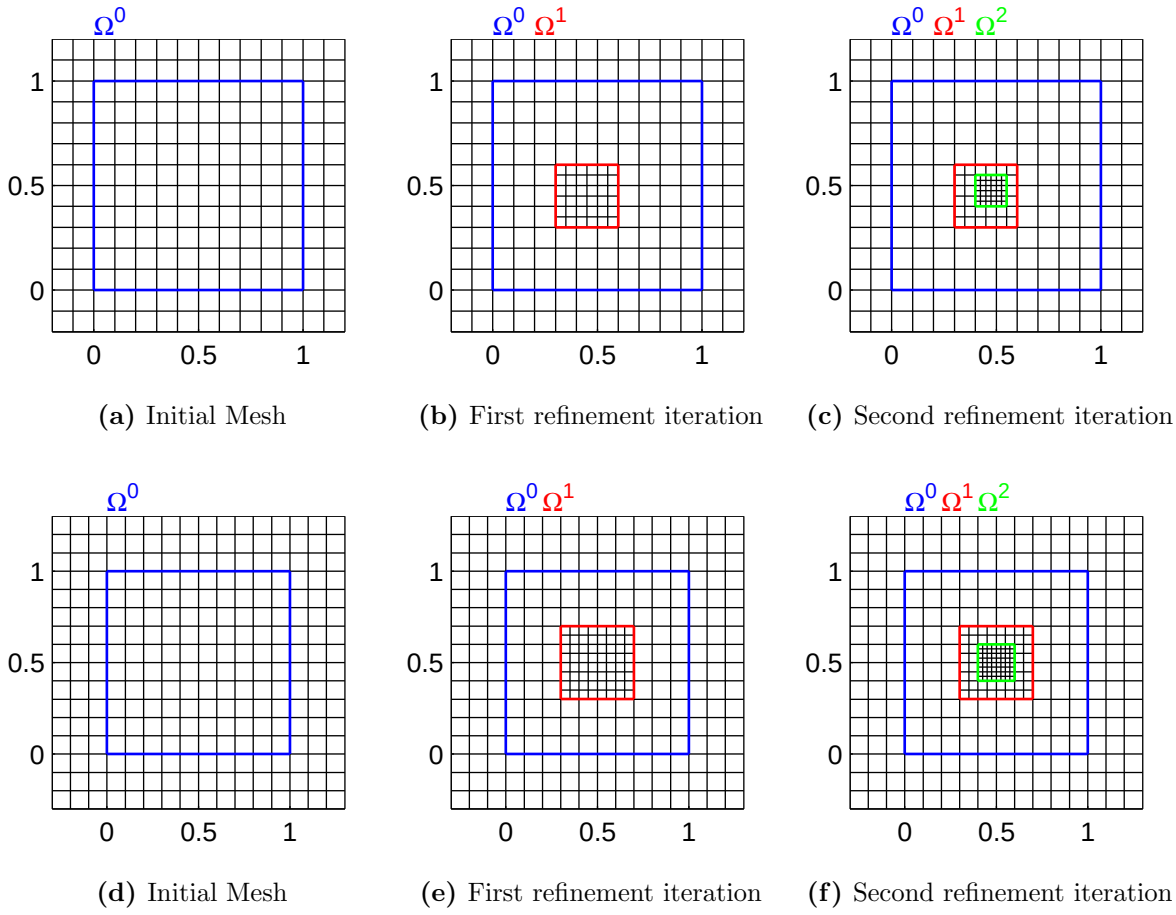


Figure 4.9: 1D Central Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 5$ at the last refinement iteration. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

4.2.3 2D Example: Central Refinement

The first of the 2D examples we present is the natural extension of the 1D cases considered above. Starting from a uniform tensor-product mesh, we performed five refinement iterations where at each step the central basis function was selected for refinement. Experiments were conducted for $p = 2, 3, 4$. As in the previous cases, for even polynomial degrees usually none of the basis functions is perfectly in the centre of the mesh; when this happened we chose to refine the lower-left function. Depending on the polynomial degree we also adjusted the knot vectors to have a ghost domain such that the initial tensor product basis constitutes a partition of unity in $\Omega^0 = [0, 1] \times [0, 1]$. Figure 4.10 shows the first three steps of the refinement process for $p = 2$ and $p = 3$.



(a) Initial Mesh **(b)** First refinement iteration **(c)** Second refinement iteration

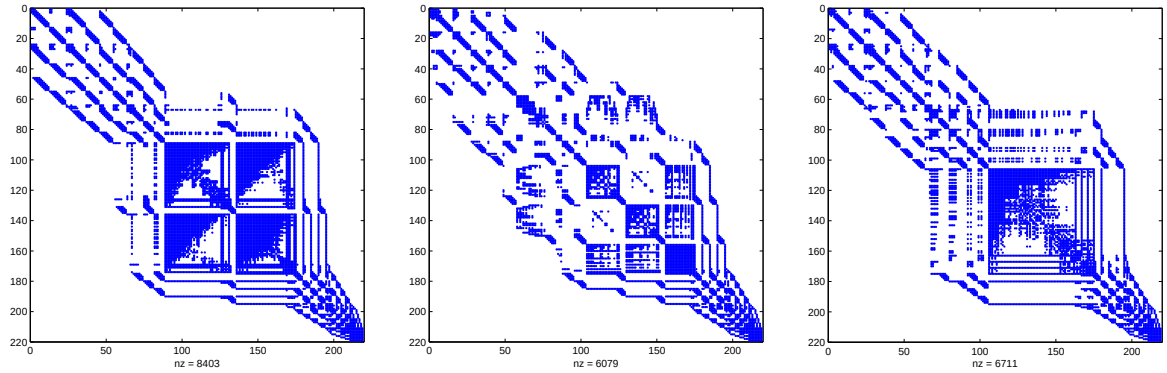
(d) Initial Mesh **(e)** First refinement iteration **(f)** Second refinement iteration

Figure 4.10: 2D Central Refinement: The first three steps of the refinement process in the cases $p = 2$ (above) and $p = 3$ (below). When four functions are equally close to the centre, the lower-left one is selected for refinement.

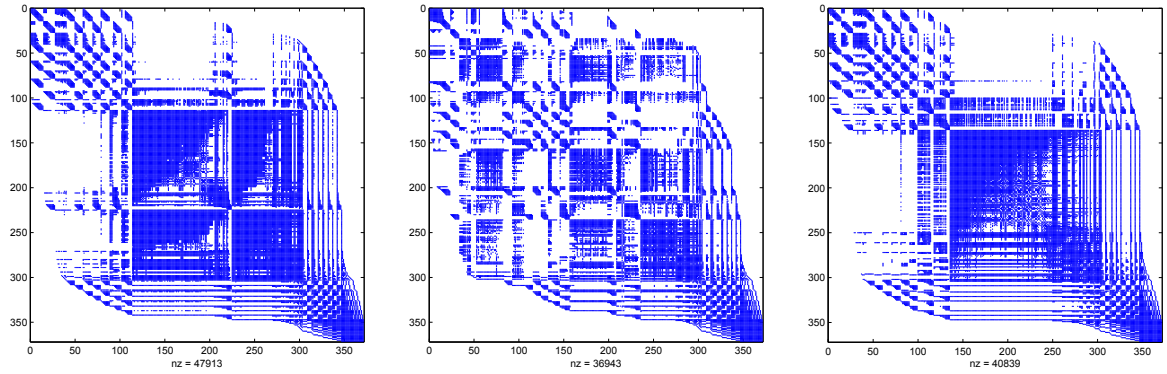
As in the previous examples, we constructed the stiffness and mass matrices using Classical Hierarchical, Truncated Hierarchical and LR B-splines basis functions on the same meshes, and compared their numerical properties.

Sparsity Figure 4.11 shows the sparsity pattern of the stiffness matrices after the fifth refinement iteration for biquadratic and biquartic basis functions. All results are reported in Table 4.4.

As expected the Classical Hierarchical basis produces significantly denser matrices due to the higher number of overlaps. While it may seem that the improvement gained by



(a) Classical Hierarchical, $p = 2$ **(b)** Truncated Hierarchical, $p = 2$ **(c)** LR B-splines, $p = 2$



(d) Classical Hierarchical, $p = 4$ **(e)** Truncated Hierarchical, $p = 4$ **(f)** LR B-splines, $p = 4$

Figure 4.11: 2D Central Refinement: The sparsity patterns of the stiffness matrices at the last (5th) refinement iteration. The Cuthill-McKee Algorithm has been applied to optimize the bandwidth. Top: $p = 2$. Bottom: $p = 4$.

using the Truncated basis is less than in the 1D case, we have to take into consideration that the refined region is very small.

Conditioning Numbers Figure 4.12 shows the plots of the conditioning numbers of the stiffness and mass matrices produced by the refinement procedure described above. As we can see, the conditioning numbers for the mass matrices are bounded for the higher degrees.

Tables 4.5 and 4.6 contain the numerical values of the conditioning numbers for $p = 2$ and 3, respectively.

p	Hier.	Trunc.	LR	H/T	H/LR
2	8403	6079	6711	138%	125%
3	23625	14909	18909	158%	125%
4	47913	36943	40839	130%	117%

Table 4.4: 2D Central Refinement: Number of non-zero elements in the stiffness matrices at the last (5th) refinement iteration. The last two columns present the ratios, rounded to the nearest percentage point.

Stiffness Matrix

Iter.	0	1	2	3	4	5
Hier.	83.6793	125.4868	142.641	152.964	159.1009	162.9791
Trunc.	83.6793	94.7517	99.6748	102.4115	103.8121	104.6173
LR	83.6793	91.0205	91.0144	91.0061	91.006	91.0105

Mass Matrix

Iter.	0	1	2	3	4	5
Hier.	2.241e+03	2.245e+03	2.245e+03	2.245e+03	5.808e+03	2.323e+04
Trunc.	2.241e+03	2.155e+03	2.154e+03	2.154e+03	5.554e+03	2.221e+04
LR	2.241e+03	2.153e+03	2.152e+03	5.981e+03	3.245e+04	1.757e+05

Table 4.5: 2D Central Refinement: The conditioning numbers for $p = 2$ in the various iterations of the central refinement. Stiffness Matrix above, Mass Matrix below.

Stiffness Matrix

Iter.	0	1	2	3	4	5
Hier.	2.323e+04	3.410e+04	3.792e+04	3.960e+04	4.043e+04	4.088e+04
Trunc.	2.323e+04	2.714e+04	2.977e+04	3.112e+04	3.186e+04	3.231e+04
LR	2.323e+04	2.416e+04	2.421e+04	2.421e+04	2.421e+04	2.421e+04

Mass Matrix

Iter.	0	1	2	3	4	5
Hier.	1.975e+06	2.016e+06	2.019e+06	2.019e+06	2.019e+06	2.019e+06
Trunc.	1.975e+06	1.837e+06	1.837e+06	1.837e+06	1.837e+06	1.837e+06
LR	1.975e+06	1.836e+06	1.836e+06	1.836e+06	1.836e+06	1.836e+06

Table 4.6: 2D Central Refinement: The conditioning numbers for $p = 3$ in the various iterations of the central refinement. Stiffness Matrix above, Mass Matrix below.

Spectrum Figure 4.13 shows the spectra of the matrices obtained from the last refinement iteration using biquadratic basis functions. While there is no substantial difference in the eigenvalue distribution produced by the three refinement methodologies, we can see how the smallest eigenvalue of the mass matrix coming from LR B-splines functions is lower than its Hierarchical counterparts. This explains the higher conditioning number for LR B-splines than the Hierarchical refinement schemes.

Increasing the polynomial degree to $p = 4$ compacts the spectrum, reducing the lower eigenvalues but also the higher ones. As in the univariate case, the smallest eigenvalues are outliers and assume almost the exact same value for all three families of splines; the difference in the magnitude of the conditioning numbers is therefore dictated by the values of the greatest eigenvalues.

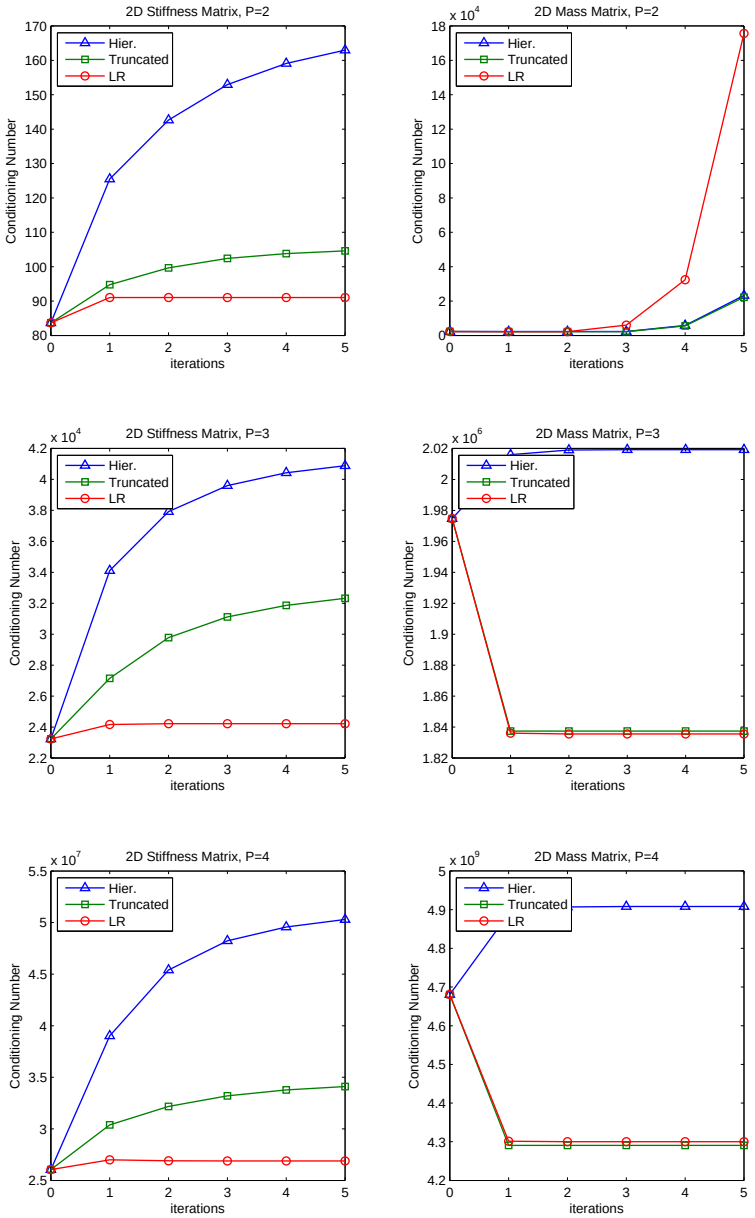


Figure 4.12: 2D Central Refinement: Graphs of the conditioning numbers of stiffness matrices (left column) and mass matrices (right column) for the bivariate central refinement. $p = 2$ (top), $p = 3$ (middle), and $p = 4$ (bottom).

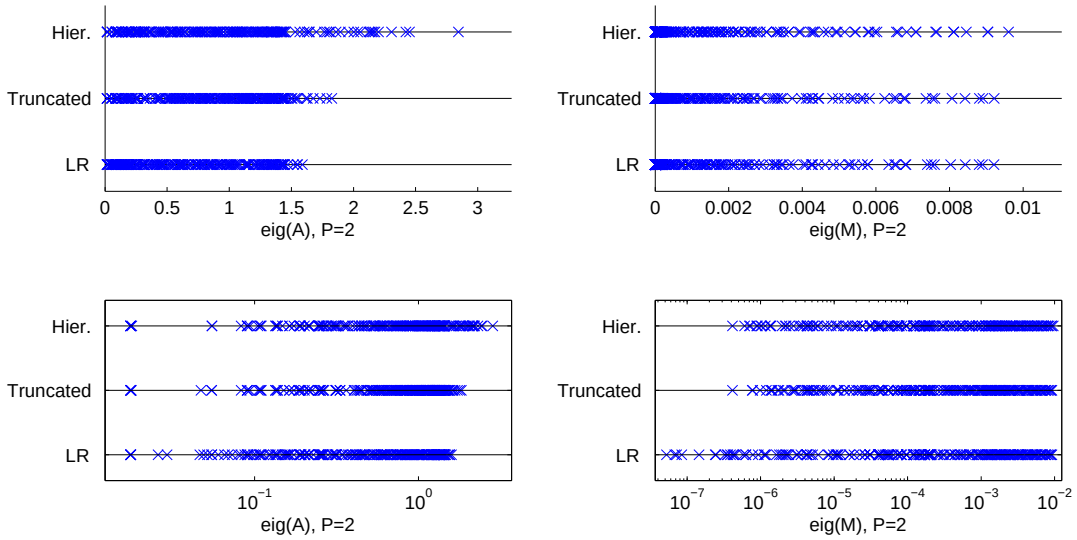


Figure 4.13: 2D Central Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 2$ at the last (5th) iteration of the central refinement, bivariate case. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

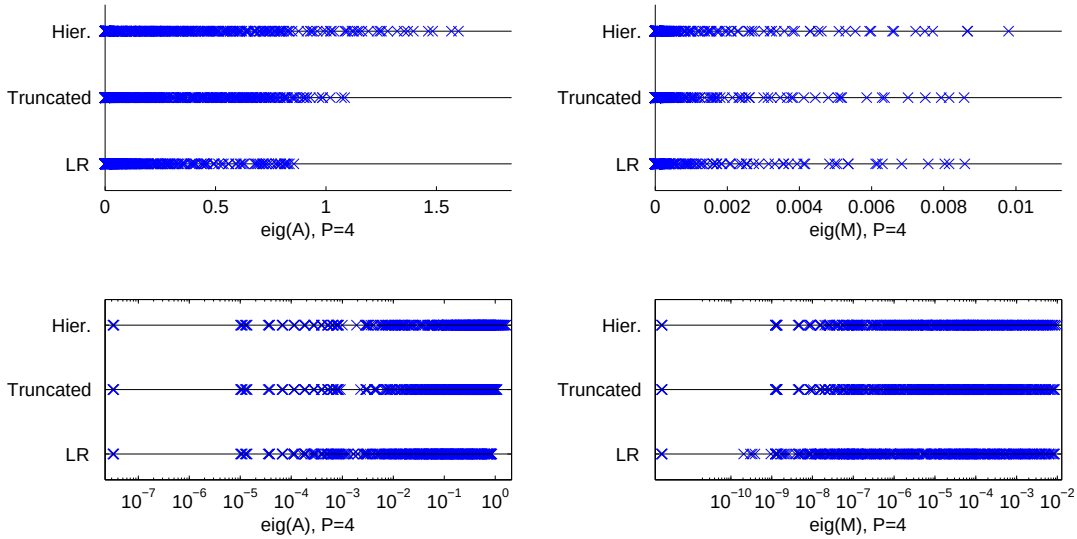
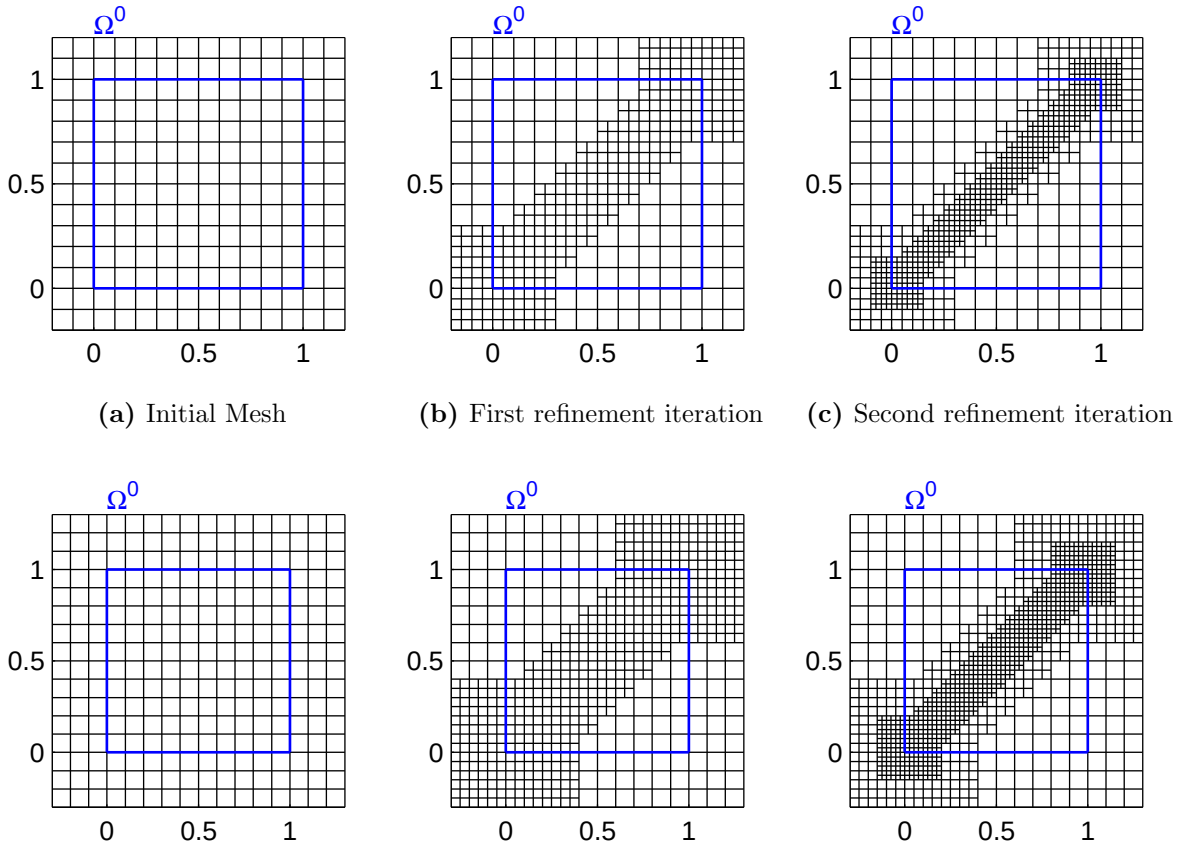


Figure 4.14: 2D Central Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 4$ at the last iteration of the central refinement, bivariate case. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

4.2.4 2D Example: Diagonal Refinement

We present here the results obtained applying a diagonal refinement. This configuration is a classical benchmark often used in publications and, since the refinement area is quite large, it provides a different point of view respect to the central refinement illustrated before.

Starting from the usual uniform tensor product mesh, we applied four refinement iterations where we refine at each step all the basis functions along the diagonal. As in the centre refinement case, we considered the polynomial degrees $p = 2, 3$, and 4. Again, the ghost domain was adjusted in order for the first tensor product basis to constitutes a partition of unity in Ω^0. Figure 4.15 shows the first three meshes in the biquadratic and bicubic cases. Note that we refined also a portion of the ghost domain: This was done in order to avoid having T-joints on the boundary of Ω^0. The integration, however, was carried out only for the elements inside Ω^0, as in the previous cases.



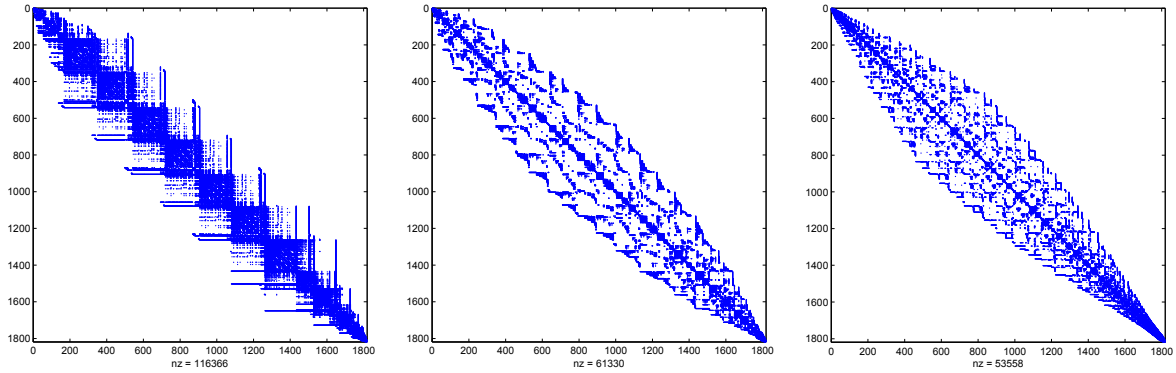
(a) Initial Mesh **(b)** First refinement iteration **(c)** Second refinement iteration

(d) Initial Mesh **(e)** First refinement iteration **(f)** Second refinement iteration

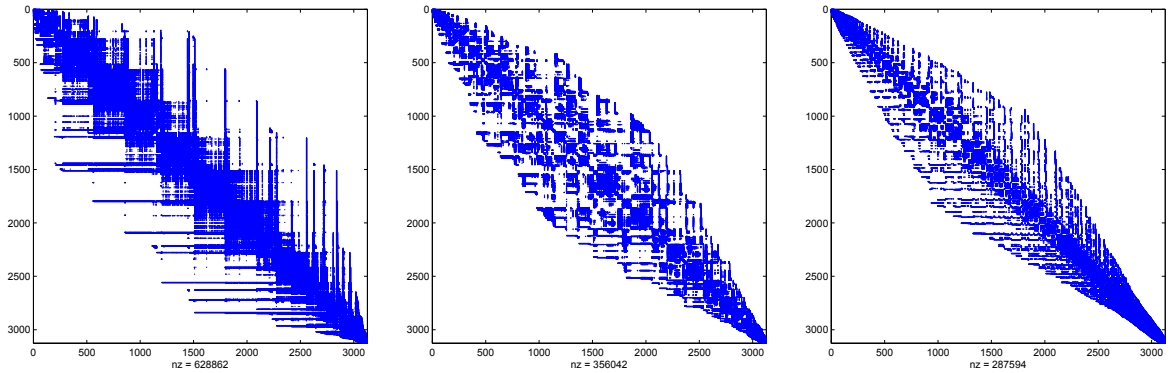
Figure 4.15: 2D Diagonal Refinement: The first three steps of the refinement process in the cases $p = 2$ (above) and $p = 3$ (below).

Sparsity Due to the extension of the refinement region, and the number of overlapping zones, we expected to see quite a difference in the sparsity pattern of the matrices produced by the different spline technologies. Figure 4.16 presents the sparsity patterns for $p = 2$ and $p = 4$. The results are presented in Table 4.7.

As we can see, due to the larger refined area the use of Truncated Hierarchical or LR



(a) Classical Hierarchical, $p = 2$ **(b)** Truncated Hierarchical, $p = 2$ **(c)** LR B-splines, $p = 2$



(d) Classical Hierarchical, $p = 4$ **(e)** Truncated Hierarchical, $p = 4$ **(f)** LR B-splines, $p = 4$

Figure 4.16: 2D Diagonal Refinement: The sparsity patterns of the stiffness matrices at the last refinement iteration for the 2D diagonal refinement. The Cuthill-McKee Algorithm has been applied to optimize the bandwidth. Top: $p = 2$. Bottom: $p = 4$.

B-splines basis functions has a huge impact on the sparsity of the matrices: The Classical Hierarchical basis produces almost twice as many non-zero elements as the Truncated basis, and more than twice those of the LR B-splines basis. It also seems that increasing the polynomial degree somewhat reduces the advantage of the Truncated basis, while the LR B-splines basis maintains the same ratio.

Conditioning Numbers Figure 4.17 shows the conditioning numbers of the stiffness and mass matrices obtained for this diagonal example. The numerical values of the conditioning numbers for the stiffness and mass matrices are presented in Table 4.8 for the biquadratic case, and Table 4.9 for the bicubic case.

p	Hier.	Trunc.	LR	H/T	H/LR
2	116366	61330	53558	190%	217%
3	304671	164039	140047	186%	218%
4	628862	356042	287594	177%	219%

Table 4.7: 2D Diagonal Refinement: Number of non-zero elements in the stiffness matrices at the last refinement iteration of the bivariate diagonal refinement. The last two columns present the ratios, rounded to the nearest percent point.

Stiffness Matrix

Iter.	0	1	2	3	4
Hier.	83.6793	139.708	169.439	225.641	318.5089
Trunc.	83.6793	97.8692	173.2625	371.711	772.2455
LR	83.6793	95.6921	163.0508	346.4521	716.4894

Mass Matrix

Iter.	0	1	2	3	4
Hier.	2.241e+03	8.783e+03	3.511e+04	1.404e+05	5.617e+05
Trunc.	2.241e+03	8.187e+03	3.275e+04	1.310e+05	5.240e+05
LR	2.241e+03	8.161e+03	3.264e+04	1.306e+05	5.223e+05

Table 4.8: 2D Diagonal Refinement: The conditioning numbers for $p = 2$ in the various iterations of the diagonal refinement. Stiffness Matrix above, Mass Matrix below.

Stiffness Matrix

Iter.	0	1	2	3	4
Hier.	2.323e+04	3.661e+04	4.366e+04	4.588e+04	4.675e+04
Trunc.	2.323e+04	2.666e+04	2.840e+04	2.927e+04	2.977e+04
LR	2.323e+04	2.522e+04	2.569e+04	2.581e+04	2.585e+04

Mass Matrix

Iter.	0	1	2	3	4
Hier.	1.975e+06	7.885e+06	3.160e+07	1.264e+08	5.057e+08
Trunc.	1.975e+06	6.876e+06	2.750e+07	1.100e+08	4.400e+08
LR	1.975e+06	6.753e+06	2.701e+07	1.080e+08	4.321e+08

Table 4.9: 2D Diagonal Refinement: The conditioning numbers for $p = 3$ in the various iterations of the diagonal refinement. Stiffness Matrix above, Mass Matrix below.

Spectrum Figures 4.18 and 4.19 present the spectrum of the stiffness and mass matrices in the cases $p = 2$ and 4, respectively. As before, the magnitude of the smallest eigenvalues is the same for all three types of basis functions considered. The value of the conditioning numbers depends therefore from the values of the highest eigenvalues, which is typically greater for the Classical Hierarchical functions.

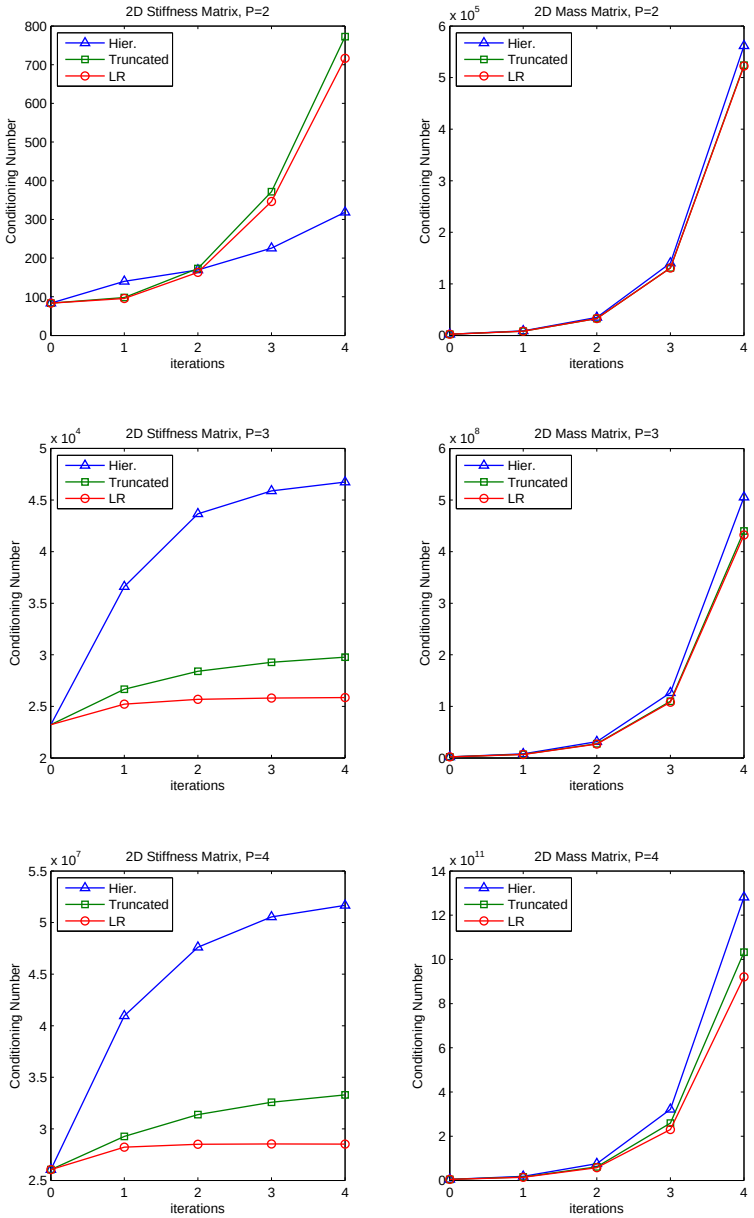


Figure 4.17: 2D Diagonal Refinement: Graphs of the conditioning numbers of stiffness matrices (left column) and mass matrices (right column) for the bivariate diagonal refinement. $p = 2$ (top), $p = 3$ (middle), and $p = 4$ (bottom).

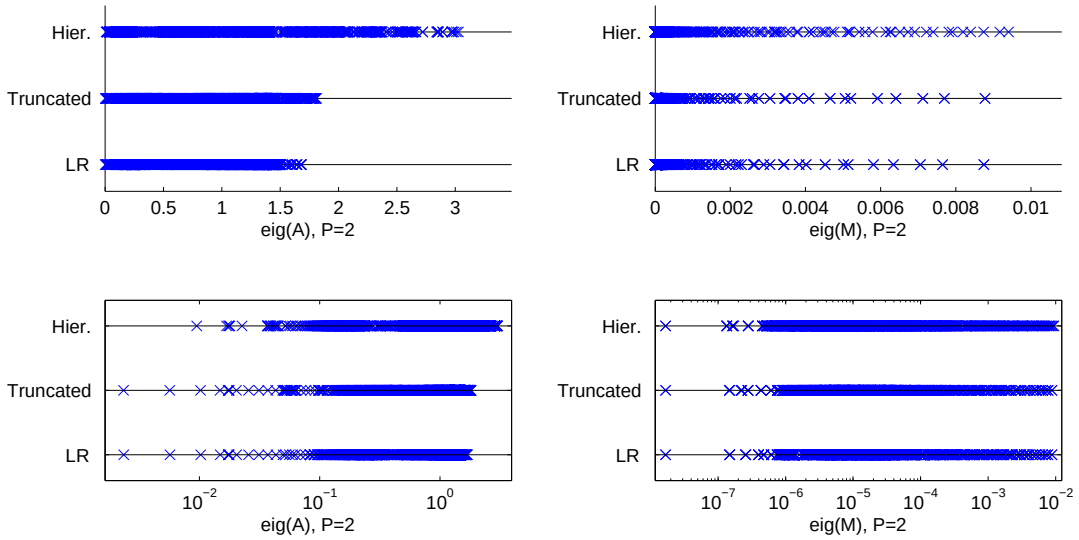


Figure 4.18: 2D Diagonal Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 2$ at the last iteration of the diagonal refinement. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

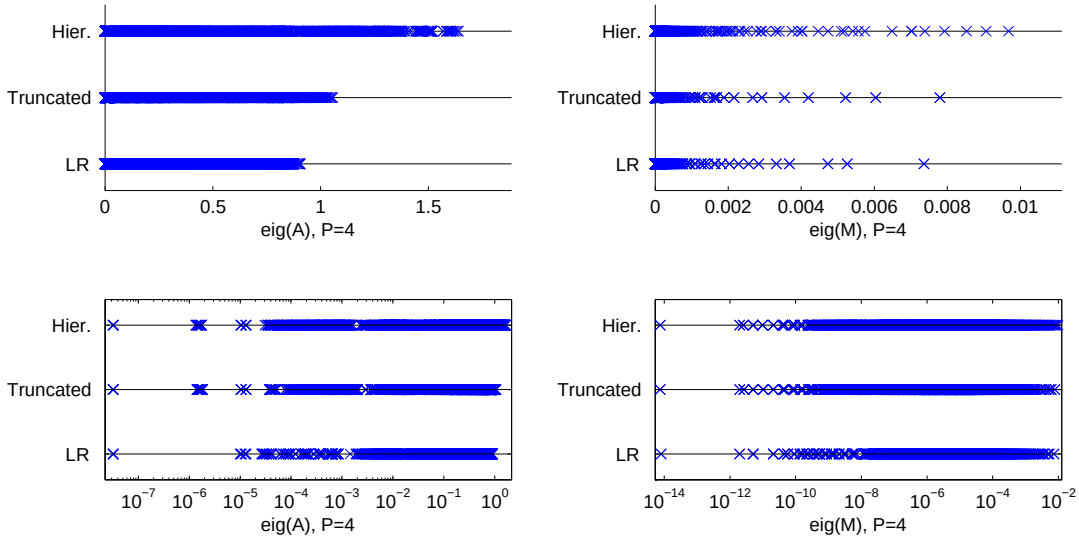


Figure 4.19: 2D Diagonal Refinement: The eigenvalues of the Stiffness Matrix (left) and Mass Matrix (right) for $p = 4$ at the last iteration of the diagonal refinement. The plots are shown on a linear scale (top) and logarithmic scale (bottom). The zero eigenvalue of the Stiffness Matrix is omitted.

5

Future work

One of the main drawbacks of the Classical Hierarchical basis described in Definition (7) is that it does not preserve partition of unity. This is a very important property of the basis functions used in Isogeometric Analysis as it is linked to the stability of the basis, seen as a relationship between a spline function f and its control points, [12]; moreover, it is also needed for consistency i.e. the need to be able to represent a constant in order to cover the nil space of the corresponding bilinear operator (the same as rigid body translations in elasticity). Due to this, it is normally preferred to use basis functions that maintain partition of unity at all stages of refinement.

As briefly stated in Chapter 3, one can easily recover the partition of unity of the Hierarchical basis by weighting the functions appropriately through the use of Equation (3.5). Doing this, however, led to some peculiar results in our experiments. Figure 5.1 shows the same conditioning numbers as in the last row of Figure 4.7, the 1D cases with central refinement where we have included also the data of the Weighted Hierarchical basis. Figure 5.2 shows the corresponding spectrum for this case. Please note that the line corresponding to the Classical Hierarchical basis is now light-blue, while the deep-blue is the Weighted Hierarchical basis.

For a 2D example, Figure 5.3 shows the same data of the bivariate central refinement presented in Figure 4.12 for the case $p = 3$. We have included also the data for the weighted Hierarchical basis.

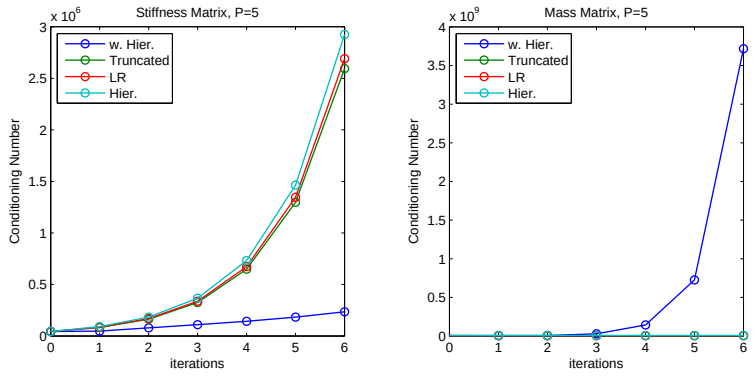


Figure 5.1: The same plot as in the last row of Figure 4.7, the 1D example with centre refinement. Here we included also the data for the Weighted Hierarchical basis, which preserves partition of unity.

As we can see, the Weighted Hierarchical basis has a drastically different behaviour

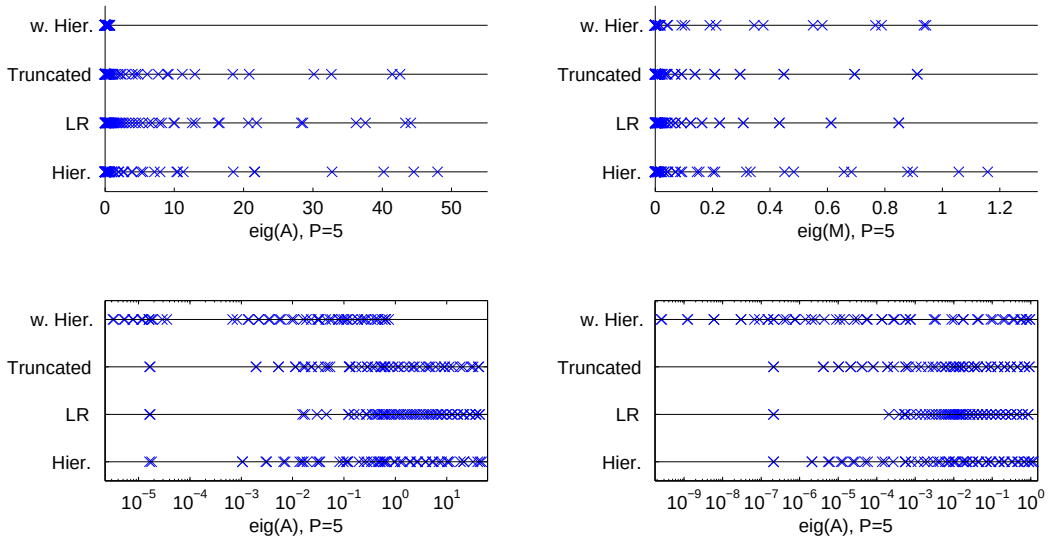


Figure 5.2: The spectrum corresponding to the 1D centre refinement with $p = 5$ previously presented in Figure 4.7. We have included also the spectrum for the Weighted Hierarchical basis.

respect to any of the other spline families considered in this paper, and more research is likely needed.

Another point we would like to investigate more is the strange behaviour we encountered in the numerical examples presented in Section 4 for the polynomial degrees $p = 2$. In almost every case, the pattern which emerged from the experiments is that Truncated Hierarchical and LR B-splines basis are very similar in performance and in particular the general trend for the conditioning number of a matrix X seems to be

$$\text{cond}(X_{LR}) \leqslant \text{cond}(X_T) < \text{cond}(X_H)$$

except for the $p = 2$ cases. We currently do not have a good argument as to why this is happening; a more careful analysis is probably required.

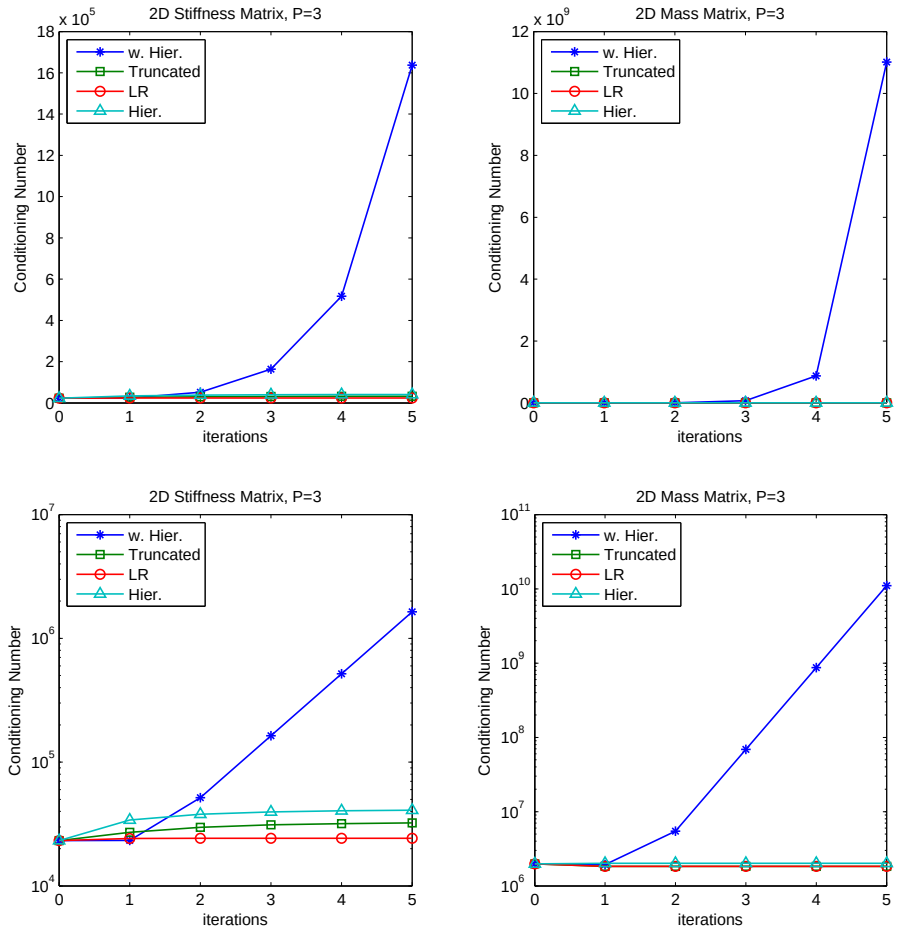


Figure 5.3: Conditioning numbers for the 2D central refinement example presented in Figure 4.12, case $p = 3$. We have included also the data for the Weighted Hierarchical Basis.

6

Conclusions

In this paper we have analysed the Classical Hierarchical, Truncated Hierarchical and LR B-splines basis on both a qualitative (more theoretical) and quantitative (more numerical) level. Regarding the qualitative differences we believe that the most important points are:

- The Classical Hierarchical basis does not constitute a partition of unity;
- For some meshes, the basis generated by the Hierarchical B-splines and the structured mesh LR B-spline refinement does indeed produce different function spaces;
- The Hierarchical and LR B-splines frameworks have different admissible meshes;
- While LR B-splines allow for more flexibility regarding the choice of refinement strategies, a formal proof for the linear independence of the resulting set of functions is still lacking.

The difference in the functions spaces is perhaps the most important point. Since both hierarchal and Truncated B-splines span the same space they are both resulting in the same discrete finite element solution, meaning that the differences in the basis functions are going to affect only the number of operations required to get to a certain precision. For LR B-splines versus Hierarchical B-splines, the situation becomes a bit more nuanced as the discrete solution itself can be different.

For the quantitative case we presented several numerical examples which have shown that there is a substantial difference between the three spline families especially for what concerns the sparsity pattern of the matrices. The Classical Hierarchical basis always produced the densest matrices, while those produced by the Truncated Hierarchical and LR B-splines were much more sparse. In particular, it seems that when the refinement region affects only a small portion of the mesh, the Truncated basis yields the best results regarding sparsity; if instead the refinement covers a large portion of the mesh, then the LR B-splines basis produces the most sparse matrices.

When it comes to the conditioning numbers, no clear and defined pattern emerged, and the results seemed very dependant on several factors: the dimension of the problem (1D vs 2D), the matrix considered (Stiffness Matrix vs Mass Matrix) and the polynomial degree. In particular, in the univariate setting we had, for the stiffness matrix,

$$\text{cond}(A_T) < \text{cond}(A_{LR}) < \text{cond}(A_H)$$

while for the mass matrix we had

$$\text{cond}(M_{LR}) < \text{cond}(M_T) < \text{cond}(M_H)$$

In the bivariate setting things are not so definite, but with the exclusion of the quadratic cases we had

$$\text{cond}(A_{LR}) < \text{cond}(A_T) < \text{cond}(A_H)$$

for the stiffness matrix, and

$$\text{cond}(M_{LR}) \approx \text{cond}(M_T) < \text{cond}(M_H)$$

for the mass matrix

On a general level we can say that the Classical Hierarchical basis performed worse than the Truncated or the LR one, while these last two frameworks yielded very similar results. Therefore, we conclude that for any application where sparsity or conditioning numbers are important quantities, one of these two refinement schemes are to be preferred.

Bibliography

[1] Y. Bazilevs, L. Beirao de 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*, 16:1031–1090, 2006.

[2] P. B. Bornemann and F. Cirak. A subdivision-based implementation of the hierarchical b-spline finite element method. *Computer Methods in Applied Mechanics and Engineering*, 2012.

[3] S. Brenner and L. R. Scott. *The mathematical theory of finite element methods.* Springer, 2005.

[4] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs. *Isogeometric Analysis, Toward Integration of CAD and FEA.* Wiley, 2009.

[5] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[6] T. Dokken, T. Lyche, and K.F. Pettersen. Polynomial splines over locally refined box-partitions. *Comput. Aided Geom. Des.*, 30(3):331–356, March 2013.

[7] 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*, 199(5-8):264 – 275, 2010.

[8] G. Farin. *Curves and Surfaces for CAGD.* Academic Press, 1990.

[9] D.R. Forsey and R.H. Bartels. Hierarchical B-spline refinement. *ACM SIGGRAPH Computer Graphics*, 22(4):205–212, 1988.

[10] K.P.S. Gahalaut and S.K. Tomar. Condition number estimates for matrices arising in the isogeometric discretizations. Technical Report 23, RICAM, 2012.

[11] C. Giannelli, B. Jüttler, and H. Speleers. THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7):485 – 498, 2012.

[12] C. Giannelli, B. Jüttler, and H. Speleers. Strongly stable bases for adaptively refined multilevel spline spaces. *Advances in Computational Mathematics*, 40:459–490, 2014.

[13] T.J.R. Hughes. *The finite element method: linear static and dynamic finite element analysis.* Prentice-hall, 1987.

[14] 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.

[15] K. J. Johannessen, T. Kvamsdal, and T. Dokken. Isogeometric analysis using LR B-splines. *Computer Methods in Applied Mechanics and Engineering*, 269:471–514, 2014.

[16] Claes Johnson. *Numerical solutions of partial differential equations by the finite element method*. Dover Publications, 2009.

[17] G. Kiss, C. Giannelli, and B. Jüttler. Algorithms and data structures for truncated hierarchical bsplines. Technical Report 14, Johannes Kepler University, 2012.

[18] R. Kraft. *Adaptive und linear unabhängige Multilevel B-Splines und ihre Anwendungen*. PhD thesis, Stuttgart, 1998.

[19] E. Kreyszig. *Introductory Functional Analysis with Applications*. Wiley, 1989.

[20] B. Mourrain. On the dimension of spline spaces on planar T-subdivisions. *Arxiv preprint arXiv:1011.1752*, November 2010.

[21] Alfio Quarteroni. *Numerical models for differential problems*. Springer, 2008.

[22] F. Remonato. Analisi isogeometrica per equazioni alle derivate parziali ellittiche. Bachelor thesis, Universitá degli Studi di Milano, 2012.

[23] D. Schillinger, L. Dedé, M.A. Scott, J.A. Evans, M.J. Borden, E. Rank, and T.J.R. Hughes. An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces. *Computer Methods in Applied Mechanics and Engineering*, 249 - 252(0):116 – 150, 2012.

[24] D. Schillinger and E. Rank. An unfitted hp-adaptive finite element method based on hierarchical B-splines for interface problems of complex geometry. *Computer Methods in Applied Mechanics and Engineering*, 200(47 - 48):3358 – 3380, 2011.

[25] M. A. Scott, X. Li, T. W. Sederberg, and T. J. R. Hughes. Local refinement of analysis-suitable T-splines. *Computer Methods in Applied Mechanics and Engineering*, 213:206–222, 2012.

[26] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche. T-spline simplification and local refinement. *ACM Transactions on Graphics*, 23(3):276–283, 2004.

[27] T.W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. *ACM Transactions on Graphics*, 22(3):477–484, 2003.

[28] A.V. Vuong, C. Giannelli, B. Jüttler, and B. Simeon. A hierarchical approach to adaptive local refinement in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 200(49-52):3554–3567, 2011.

[29] P. Wang, J. Xu, J. Deng, and F. Chen. Adaptive isogeometric analysis using rational PHT-splines. *Computer-Aided Design*, 43:1438–1448, 2011.

[30] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design.* Morgan Kaufmann Publishers, 2002.