TUM School of Engineering and Design

The multi-level Bézier extraction for hierarchical local refinement of trimmed isogeometric finite cell analyses

Davide D'Angella, M.Sc.(hons)

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Kai-Uwe Bletzinger
Prüfer der Dissertation:	1. Prof. Dr. Ernst Rank
	2. Prof. Dr. Andreas Schröder
	3. Prof. Alessandro Reali, Ph.D.

Die Dissertation wurde am 02.08.2021 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 27.01.2022 angenommen.

Abstract

The finite cell method (FCM) and isogeometric analysis (IGA) promise to improve the interoperability of computer-aided design (CAD) and computer-aided engineering (CAE). In addition, local refinement can be used to efficiently resolve small-scale features. The computational framework composed of these three technologies constitutes a promising approach suitable for sophisticated engineering designs.

This combination presents some unresolved issues, such as the difficulty of introducing the framework into existing finite element software, the implementation efficiency, and the inapplicability of some standard procedures (e.g., the calculation of reaction forces).

The goal of this thesis is to develop several technologies that address these problems. In particular, a multi-level Bézier extraction is proposed to facilitate the integration of locally-refined isogeometric analysis into existing implementations. The approach introduces a set of basis functions independent of the refinement level and common to each element. Algorithms for mesh refinement and coarsening are presented, and their complexity is analyzed. These algorithms operate only through univariate operators to leverage the tensor-product structure. This approach is combined with the sum factorization in a framework suitable for matrix-free iterative solvers. It is shown how local refinement can mitigate issues specific to the FCM and trimming. Namely, the unphysical coupling between the sides of a thin hole and the overconstraining induced by weak boundary conditions are discussed. Finally, a method is developed to compute the conservative reactions on trimmed locally-refined meshes subject to weak boundary conditions. The technique is explained by a variational argument and applied to the trimmed FCM with basis functions that are not defined on "nodes". The proposed approach shows the superconvergence characteristics typical of the classic procedure for untrimmed meshes and nodal shape functions.

The applicability of the proposed methods is demonstrated through various numerical examples. These include trimmed linear Kirchhoff-Love shells defined by CAD models and a three-dimensional geometry defined by a detailed STL file with smooth wavy boundaries and several internal cavities.

This work proposes several techniques that favor the applicability and efficiency of locally-refined isogeometric finite cell analyses, streamlining the design-through-analysis process.

Zusammenfassung

Die Finite-Zellen-Methode (FCM) und die isogeometrische Analyse (IGA) versprechen, die Interoperabilität von Computer-Aided Design (CAD) und Computer-Aided Engineering (CAE) zu verbessern. Darüber hinaus können kleinskalige Merkmale mithilfe lokaler Verfeinerung effizient aufgelöst werden. Ein aus diesen drei Technologien zusammengesetztes Framework stellt einen vielversprechenden Ansatz dar, der sich für komplexe Konstruktionen eignet. Diese Kombination wirft jedoch einige Probleme auf, wie z. B. die Schwierigkeit der Einbindung des Frameworks in bestehende Finite-Elemente-Software, die Effizienz der Implementierung und die Unanwendbarkeit einiger Standardverfahren (z. B. zur Berechnung von Reaktionskräften).

Ziel dieser Arbeit ist es, mehrere Technologien zu entwickeln, die diese Probleme adressieren. Insbesondere wird eine Multi-Level Bézier-Extraktion vorgeschlagen, um die Integration der lokal verfeinerten isogeometrischen Analyse in bestehende Implementierungen zu erleichtern. Die Methode führt einen Satz von Basisfunktionen ein, die für jedes Element gleich sind, unabhängig von der lokal verfeinerten Basis. Es werden Algorithmen zur Netzverfeinerung und -vergröberung vorgestellt und ihre Komplexität analysiert. Diese Algorithmen basieren ausschließlich auf univariaten Operatoren, um die Tensorproduktstruktur zu nutzen. Zudem werden diese Algorithmen mit der Summenfaktorisierung in einer Weise kombiniert, die für matrixfreie iterative Löser geeignet ist. Es wird gezeigt, wie die lokale Verfeinerung spezifische Probleme der FCM und des Trimmens entschärfen kann. Diskutiert werden die unphysikalische Kopplung zwischen den Seiten eines schmalen Lochs und die durch schwache Randbedingungen induzierte übermäßige Restriktion der Lösung. Schließlich wird eine Methode entwickelt, um die Reaktionen auf getrimmten lokal verfeinerten Netzen mit schwachen Randbedingungen zu berechnen. Die Technik wird durch ein Variationsargument erklärt und auf die FCM angewendet, unter Verwendung getrimmter Basisfunktionen, die nicht knotenbasiert sind. Dieser Ansatz zeigt die Superkonvergenzeigenschaften, die für das klassische Verfahren mit ungetrimmten Netzen und knotenbasierten Formfunktionen typisch sind.

Die Anwendbarkeit der vorgeschlagenen Methoden wird anhand verschiedener numerischer Beispiele demonstriert, einschließlich getrimmter linearer Kirchhoff-Love-Schalen, die durch CAD-Modelle definiert sind, und eines dreidimensionalen linearen Wärmeleitungsproblems, bei dem die Geometrie durch eine detaillierte STL-Datei mit glatten wellenförmigen Rändern und mehreren internen Hohlräumen definiert ist.

In dieser Arbeit werden mehrere Techniken vorgeschlagen, welche die Anwendbarkeit und Effizienz lokal verfeinerter isogeometrischer Finite-Zellen-Analysen verbessern.

Acknowledgements

The journey to this dissertation has been a period of fundamental personal change and growth in many aspects. I have been helped by many along the way, and I would like to take a moment to thank them.

I would like to warmly thank my doctoral supervisors (in alphabetical order), Prof. Ernst Rank and Prof. Alessandro Reali, for their constant support, guidance, and (scientific and non-scientific) advice. They always showed a great deal of trust that gave me the freedom to follow the research direction of my preference, creating an enjoyable and productive environment.

I feel that Prof. Rank always believed in me. Since we first met at the Ferienakademie, he offered me to join his group as a research assistant and suggested I take a Ph.D. position later. I would not have written this dissertation if it were not for him.

Prof. Reali or, as I usually call him, Ale, created a very informal relationship from the beginning that made me able to speak to him about virtually anything. I could get extensive scientific advice from him and his network and enjoy the time together. Such an environment made me benefit more from his supervision and also enjoy this journey. He also created the opportunity to visit the University of Texas at Austin, an experience that I will never forget.

My special thanks go to Dr. Stefan Kollmannsberger, who always advised and participated in my research. He continuously transmitted a lot of generosity, humanity, and warmth. He is a crucial ingredient in creating a good team and a friendly atmosphere.

My special appreciation and gratitude go to Prof. Thomas J.R. Hughes, whom I visited at the University of Texas at Austin. The stimulating scientific input and constant supervision significantly contributed to my favorite part of this dissertation. I enjoyed a lot his hospitality and my stay in Austin.

I would also like to thank Prof. Giancarlo Sangalli for several scientific discussions on various topics. I highly appreciate his feedback and interest in my work.

Thanks to Prof. Andreas Schröder for being part of my Ph.D. committee after having already supervised and examined my masters thesis some years ago.

I also thank Prof. Bletzinger for chairing the examination committee. His humor created a very comfortable atmosphere on the day of my Ph.D. thesis defense.

It was great to share this journey with my colleagues and friends at the Chair for Computation in Engineering, Chair of Computational Modeling and Simulation and TU Hamburg: Nils, Mohamed, Chris, Bobby, Lisa, John, Philipp, Nina, László, Ralf, Alex P., Alex B., Alex M., Paul, Katrin, Nevena, Jing, Benjamin, Jimmy, Tim, Jan, Ann, Vijaya, Leon, Oz, Simon, Tino, Robert, Ali, Prof. André Borrmann, Vasiliki, Hanne, Lars, André, Mahan, and Wadhah. Thanks for bearing with me. I have tested the patience of many of you several times :)

Special thanks to my new adventure companions, Nina and László, for being so close over the last year. Nina pushed and helped me a lot in finalizing this dissertation. Also, thank you for the happiness provided in the form of cakes and coffee :)

I had a great time also at my "second university" in Pavia, where I shared several months with the CompMech group: Alice, Lorenzo, Giorgione, Simone, Marghe, Alberto, Giulia, Vale, Rodrigo, Alex, Michele, Laura, Kayal, Max, Alessia, Guillermo, Isabella,

Stefania, Gianluca, Valeria, and Sonia. Thank you for the friendly hospitality and mood.

It was great to share my stay in Austin with some of the students and visitors of Prof. Thomas J.R. Hughes: René, Jeff, Hiroshi, Kendrick, Baidoo, Deepesh (it was fun to prove you that I really cannot play ping pong), Lulin (I had nice time sharing the office and having lunch with you everyday), Michael and Tyler (it was nice to see you training for the pizza challenge, kayaking the Colorado river, talking for hours and hours about the football rules, and hanging out in Austin and Munich).

Special thanks to Luca and Massimo: we have always had a ton of fun at all conferences we attended together, be it Cetraro or New York. Thank you for being a magnet for crazy episodes like having questionable bets with some professors at night, making an opera singer sing at a conference dinner, or intense karaokes. Of course, I was the normal one among the three of us :D

Thanks to Ondina for constantly transmitting a good mood and making fun of Luca with me all the time.

Thanks to my international friends in Munich for all the good times, greek food, pizzas, weddings, and fun: Omero, Sha, Christina, Odyx, Alex, Silva, and Dennis.

Thanks to my good old friends from lovely Cavallino–Treporti: Lola, Bea, Carry, Elisa, Flaviano, Giuly, il Salso, il Don, and Anna. They always give me the best mood and endless laughs when I am back "home". A special thanks to Alice and Nicoló, who cooked with me through video calls nearly every Friday over the last years of the Ph.D. You just fill my life with life, no matter how far apart we can be.

My appreciation also goes out to my family, who has given me all the freedom and support to make my own choices.

A special word of thanks to the late Prof. Massimo Tarallo. He was my most influential professor who transmitted a lot of knowledge and a mathematical way of thinking.

And my warmest and loveliest thanks go to my Principessa. Sometimes, it has been a rocky ride, but you were unconditionally there for me, armed with a lot of patience and, most of all, love (and further necessary patience). You are my biggest support and strength.

Davide D'Angella
Munich, February 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Modern production technologies allow today's engineering designs to be less constrained by the manufacturing process and primarily driven by functionality. As a result, simulation engineers must deal with objects with increasingly sophisticated shapes defined by nature-inspired parts, overhangs, dents, irregular holes, smooth curved boundaries, or multiple porous regions. The transformation of these complex computer-aided designs (CAD) into models suitable for analysis represents one of the most time-consuming activities in computer-aided engineering (CAE).

The finite cell method (FCM) and isogeometric analysis (IGA) were introduced to improve interoperability between CAD and CAE. The former method does not require a geometry-conforming computational mesh, eliminating the time-consuming and error-prone mesh generation step. The latter uses the CAD description directly as a computational mesh, unifying the models used for design and analysis.

An additional difficulty is that complex geometries often lead to small-scale solution features, such as stress concentrations or singularities. The accuracy can be efficiently increased by reducing the element size only in these regions of interest. The mesh-refinement process is considerably eased by allowing so-called hanging nodes, i.e., nodes belonging to one element but not to one of its neighbors. This approach is usually referred to as local refinement. This work considers a straightforward refinement formulation based on multiple levels defining increasingly finer basis functions. This approach is referred to as multi-level or hierarchical refinement.

The finite cell method, isogeometric analysis, and local refinement can be combined, creating a powerful and accurate framework that tightens the design-through-analysis loop. Although this combination has been successfully applied in several examples, some crucial issues remain unresolved:

- The implementation of local refinement is unconventional, making it challenging to implement this approach in existing software.
- Naive implementations of local refinement may be inefficient.

- The framework is unconventional, making some standard procedures inapplicable (e.g., the calculation of reaction forces).

1.2 Objectives

This thesis aims to develop several supporting technologies to alleviate the above problems, and to favor the use of the described computational framework in modern engineering practice. In particular, the objectives of this work are to:

- Propose a strategy that brings the multi-level local refinement closer to traditional finite element implementations.

  This goal requires aligning the hierarchical definition of element shape functions with the classical definition of standard basis functions common to all elements.
- Formulate several algorithms to efficiently perform common mesh-adaptivity operations in combination with the multi-level definition of basis functions. These operations should include the shape function evaluation, projection of degrees of freedom for refinement and coarsening, and evaluation of matrix-vector multiplications for matrix-free iterative solvers.

  The goal is to define procedures for performing these operations and study their cost as a function of the number of spatial dimensions, the number of refinement levels, and the polynomial order of the basis functions.
- Provide algorithms and theoretical understanding in the calculation of reaction forces using the described computational framework.

  The absence of the standard concept of "nodes" on the domain boundary and the application of boundary conditions in a "weak sense" preclude the use of standard procedures. The goal is to formulate suitable algorithms for the computation of reaction forces retaining the same convergence rates as the classical approach.

1.3 Outline

The coming chapters formally define the computational framework and address the objectives described above. Their content can be summarized as follows:

- **Chapter 2**[ab] defines the B-splines and NURBS basis functions, a common ingredient in CAD geometry descriptions. These functions are also used for the numerical analysis, following the isogeometric methodology. The Kronecker product and its properties are briefly reviewed. Successively, this chapter introduces the Bézier extraction: a technique that facilitates the implementation of IGA in existing software (without local refinement).

[a] This chapter is based on D'Angella et al. [2018]. The main scientific research as well as the textual elaboration of the publication was performed by the author of this work. Part of the text of this chapter is taken in an adjusted version from D'Angella et al. [2018].

[b] This chapter is based on D'Angella and Reali [2020]. The main scientific research as well as the textual elaboration of the publication was performed by the author of this work. Part of the text of this chapter is taken in an adjusted version from D'Angella and Reali [2020].

- **Chapter 3**[ab] defines the considered multi-level local refinement suitable for isogeometric discretizations. In particular, the hierarchical B-splines and their truncated variant are discussed.
- **Chapter 4**[a] proposes an extension of the Bézier extraction to (truncated) hierarchical B-splines. The core idea is first introduced for one-dimensional geometries and then extended to higher-dimensional domains. The approach is discussed at both global and element-local levels. The generalization of the method to other refinement schemes is finally outlined.
- **Chapter 5**[ab] proposes an approach for evaluating the shape functions and projecting the degrees of freedom for refinement and coarsening. A modified Bézier projection scheme is proposed to leverage the local tensor-product structure. The algorithms are tested in a transient and in a non-linear three-dimensional example.
- **Chapter 6**[cd] discusses the use of local refinement in combination with IGA and FCM to mitigate two issues: the unphysical coupling across thin holes and the overconstraining induced by the weak boundary conditions. The application of the computational framework is illustrated in two shell numerical examples.
- **Chapter 7** presents the theoretical and algorithmic foundations for the computation of reaction forces on immersed boundaries subject to weak boundary conditions. The proposed approach is also suitable for basis functions that are not based on the concept of "nodes" and do not form a partition of unity. The method is tested on a simple 2D benchmark, a shell example, and a three-dimensional complex geometry defined by smooth wavy boundaries and several internal cavities.
- **Chapter 8** combines the algorithms presented in Chapter 5 into a framework suitable for matrix-free iterative solvers. The approach uses the multi-level Bézier extraction and moment-fitting quadrature to exploit the present tensor structure. The application of penalty boundary conditions is finally discussed.
- **Chapter 9**[bc] summarizes the main results and outlines possible future research directions.

Part of the research presented in this thesis has been published in several scientific papers. The proposed Bézier extraction for multi-level refinement was first presented in D'Angella et al. [2018]. The related algorithms were introduced in D'Angella et al. [2018]; D'Angella and Reali [2020]. The use of local refinement to mitigate the unphysical coupling and weak overconstraining was jointly published in Coradello et al. [2020b], where the main scientific research and the textual elaboration of the publication have been equally developed in close collaboration between the first two authors, Luca Coradello and Davide D'Angella.

[c] This chapter is based on Coradello et al. [2020a]. The main scientific research and the textual elaboration of the publication have been equally developed in close collaboration between the first two authors Luca Coradello and Davide D'Angella. Parts of Coradello et al. [2020a] has been included in the dissertations of both first two authors.

[d] Part of the text of this chapter is taken in an adjusted version from Kollmannsberger et al. [2020], a publication co-authored by the author of this work.

Chapter 2

Isogeometric analysis

This chapter reviews the essentials of geometric modeling through NURBS and its use in isogeometric analysis [Cottrell et al., 2009; Hughes et al., 2005]. These notions are explained following closely Cottrell et al. [2009]; Lyche and Morken [2008]; Piegl and Tiller [1995], establishing the context within which the main contributions of this dissertation are developed. The chapter concludes by introducing the Bézier extraction [Borden et al., 2011]: a concept that is generalized for hierarchical refinement in a new contribution elaborated in Chapter 4.

2.1 B-splines

The univariate B-spline functions of degree p are defined through a non-decreasing sequence of real numbers

$$\boldsymbol{\Xi} = (\xi_1, ..., \xi_{m+p+1}), \qquad \xi_i \leq \xi_{i+1}, \qquad i = 1...m+p. \tag{2.1}$$

The quantities ξ_i are called knot values, or knots, and $\boldsymbol{\Xi}$ is referred to as knot vector. Note that the same knot value can be repeated multiple times. A knot value is said to have a multiplicity equal to k when repeated for a total number k of times. The interval $[\xi_i, \xi_{i+1})$ is called the ith knot span, and it has zero length when $\xi_i = \xi_{i+1}$. The vector of distinct knot values

$$\breve{\boldsymbol{\Xi}} = (\breve{\xi}_1, ..., \breve{\xi}_{\breve{m}+1}), \tag{2.2}$$

is obtained from $\boldsymbol{\Xi}$ by reducing all multiplicities to one. Namely, $\breve{\boldsymbol{\Xi}}$ represents the non-degenerate knots spans.

The knot vector $\boldsymbol{\Xi}$ defines m B-spline functions $\{N_i^p\}_{i=1...m}$ through the Cox-de Boor recursion formula

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi). \tag{2.3}$$

The recursion terminates for B-splines N_i^0 of degree $p = 0$, i.e., step functions on the half-open knot interval $[\xi_i, \xi_{i+1})$

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

When the quotient 0/0 occurs in Equation (2.3), it is defined to be equal to zero.

Note that from their definition, the B-splines are piecewise polynomials defined on the whole real line, and they assume non-zero values only within the interval given by the extremal knot values (ξ_1, ξ_{m+p+1}). If a knot value is repeated k times, $k \geq 1$, then the B-spline functions have $p-k$ continuous derivatives at that location, while they are infinitely differentiable everywhere else. In the following, only knot vectors where the first and last knot are repeated $p+1$ times are considered

$$\boldsymbol{\Xi} = (\underbrace{a, ..., a}_{p+1}, \xi_{p+2}, ..., \xi_m, \underbrace{b, ...b}_{p+1}). \tag{2.4}$$

Such knot vectors are referred to as open or clamped. Without loss of generality, it is defined $(a, b) = (-1, 1)$.

A B-spline curve $\boldsymbol{F} : [-1, 1] \to \mathbb{R}^{m_g}$ in an m_g-dimensional space can be defined by some coefficients $\{\boldsymbol{P}_i\}_{i=1...m}$, $\boldsymbol{P}_i \in \mathbb{R}^{m_g}$ referred to as control points. The value $\boldsymbol{F}(\xi)$ for a parametric location $\xi \in \mathbb{R}$ is defined by the linear combination

$$\boldsymbol{F}(\xi) = \sum_{i}^{m} N_i^p(\xi) \boldsymbol{P}_i.$$

In the following chapters, such linear combination will be often expressed in matrix notation as

$$\boldsymbol{F}(\xi) = \boldsymbol{P}^\top \, \boldsymbol{N}^p(\xi),$$

where

$$\boldsymbol{N}^p(\xi) = \left(N_1^p(\xi), ..., N_m^p(\xi)\right)^\top, \tag{2.5}$$

$$P_{ij} = \left[\boldsymbol{P}_i\right]_j. \tag{2.6}$$

2.2 Non-uniform rational B-splines

The B-spline functions defined in the previous section can be generalized to NURBS, allowing to represent a broader class of geometric objects such as conic sections. NURBS are nowadays the de-facto industry standard in the design and exchange of digital geometries.

The pth-degree NURBS basis functions $\bar{\boldsymbol{R}}^p(\xi)$ are defined by an open knot vector $\boldsymbol{\Xi} = (\xi_1, ..., \xi_{m+p+1})$ and some weights $\boldsymbol{w} \in \mathbb{R}^m$. Specifically,

$$\bar{R}_i^p(\xi) = \frac{w_i}{\boldsymbol{w}^\top \boldsymbol{N}^p(\xi)} N_i^p(\xi), \tag{2.7}$$

where $\boldsymbol{N}^p$ is composed of the B-splines defined by the knot vector $\boldsymbol{\Xi}$, as described in the previous section.

A NURBS curve $\boldsymbol{F} : [-1, 1] \to \mathbb{R}^{m_g}$ is constructed by a linear combination of NURBS basis functions

$$\boldsymbol{F}(\xi) = \boldsymbol{P}^\top \, \bar{\boldsymbol{R}}^p(\xi),$$

for some $m \times m_g$ coefficient matrix $\boldsymbol{P}$, indicated as $\boldsymbol{P} \in \mathbb{R}^{m \times m_g}$.

Note that the NURBS basis functions are a generalization of the standard B-splines, as their definition coincides when the weights are all equal to one (or any other constant value). Moreover, the following properties hold

$$\begin{aligned}
\bar{R}_i^p(\xi) &\geq 0, && \forall i, p, \text{ and } \xi \in [-1, 1] && \text{(non-negativity)},\\
\bar{R}_i^p(\xi) &= 0, && \forall \xi \notin [\xi_i, \xi_{i+p+1}] && \text{(local support)},\\
\bar{R}_1^p(-1) &= \bar{R}_m^p(1) = 1, && \forall p && \text{(interpolatory endpoints)},\\
\sum_{i=1}^{m} \bar{R}_i^p(\xi) &= 1, && \forall \xi \in [-1, 1] && \text{(partition of unity)}.
\end{aligned}$$

See Piegl and Tiller [1995] for a comprehensive review.

An efficient way to represent NURBS curves is to use homogeneous coordinates. The idea is to represent a rational geometry in m_g coordinates as a B-spline geometry in an $(m_g + 1)$-dimensional space. Given some control points $\{\boldsymbol{P}_i\}$ and non-zero weights $\{w_i\}$, the curve $\boldsymbol{F}(\xi) = \sum_i \boldsymbol{P}_i \, \bar{R}_i^p(\xi)$ can be transformed into homogeneous coordinates $\mathring{\boldsymbol{P}}_i$ as

$$\mathring{\boldsymbol{P}}_i = (w_i \boldsymbol{P}_i, w_i)^\top. \tag{2.8}$$

For example, in three dimensions, $\boldsymbol{P}_i$ has three coordinates $\boldsymbol{P}_i = (x_i, y_i, z_i)^\top$, and its homogeneous representation reads

$$\mathring{\boldsymbol{P}}_i = (w_i x_i, w_i y_i, w_i z_i, w_i)^\top. \tag{2.9}$$

The original coefficients $\boldsymbol{P}_i$ can be recovered through the following map

$$\begin{aligned}
\Psi\left(\mathring{\boldsymbol{P}}_i\right) &= \Psi\left((x_i, y_i, z_i, w_i)^\top\right)\\
&= \begin{cases} \left(\frac{x_i}{w_i}, \frac{y_i}{w_i}, \frac{z_i}{w_i}\right)^\top & \text{if } w_i \neq 0\\ (x_i, y_i, z_i)^\top & \text{otherwise.} \end{cases}
\end{aligned}$$

Using the coefficients in homogeneous coordinates, the four-dimensional representation of $\boldsymbol{F}$ becomes

$$\mathring{\boldsymbol{F}}(\xi) = \sum_i \mathring{\boldsymbol{P}}_i \, N_i^p(\xi). \tag{2.10}$$

Note that $\mathring{\boldsymbol{F}}$ is defined as a linear combination of B-splines N_i^p instead of rational functions $\bar{R}_i^p$.

The rational curve $\boldsymbol{F}$ can be obtained from $\mathring{\boldsymbol{F}}$ through the map $\Psi(\cdot)$ as

$$\begin{aligned}\Psi\left(\mathring{\boldsymbol{F}}\right) &= \Psi\left(\sum_i \mathring{\boldsymbol{P}}_i\, N_i^p\right)\\ &= \sum_i \boldsymbol{P}_i\, \frac{w_i}{\boldsymbol{w}^\top \boldsymbol{N}^p\left(\xi\right)}\, N_i^p\left(\xi\right)\\ &= \sum_i \boldsymbol{P}_i\, \bar{R}_i^p(\xi)\\ &= \boldsymbol{F}.\end{aligned}$$

When there is no ambiguity, the superscript p is dropped in the following sections to ease the notation.

2.3 Multivariate geometries

Given a multi-index $\boldsymbol{i} = (i_1, ..., i_D)$, the D-variate B-spline function $N_{\boldsymbol{i}}$ can be defined as the tensor-product of univariate B-spline functions. In particular, let

$${}^d\Xi = \left({}^d\xi_1, ..., {}^d\xi_{{}^dm+{}^dp+1}\right), \qquad 1 \le d \le D, \tag{2.11}$$

be the knot vector in the dth parametric direction. As explained in the previous section, such a knot vector defines the B-spline functions ${}^dN_{i_d}$, $i_d = 1, ..., {}^dm$, of degree ${}^dp$.

The D-variate B-spline function $N_{\boldsymbol{i}}(\boldsymbol{\xi})$ at a location $\boldsymbol{\xi} = (\xi_1, ..., \xi_D)$ is defined as the tensor-product

$$N_{\boldsymbol{i}}(\boldsymbol{\xi}) = \prod_{d=1}^{D} {}^dN_{i_d}\left(\xi_d\right). \tag{2.12}$$

Multivariate geometric objects can be obtained as a linear combination of the basis functions $N_{\boldsymbol{i}}$. In particular, given some coefficients $\boldsymbol{P}_{\boldsymbol{i}}$ and the parametric domain $\hat{\Omega} = [-1, 1]^D$, a D-variate B-spline geometry $\boldsymbol{F}(\boldsymbol{\xi}) : \hat{\Omega} \to \mathbb{R}^{m_g}$ is obtained as

$$\boldsymbol{F}(\boldsymbol{\xi}) = \sum_{\boldsymbol{i}} N_{\boldsymbol{i}}(\boldsymbol{\xi})\, \boldsymbol{P}_{\boldsymbol{i}}. \tag{2.13}$$

Multivariate NURBS functions $\{\bar{R}_{\boldsymbol{i}}(\boldsymbol{\xi})\}$ with weights $\{w_{\boldsymbol{i}}\}$ are defined as

$$\bar{R}_{\boldsymbol{i}}(\boldsymbol{\xi}) = \frac{N_{\boldsymbol{i}}(\boldsymbol{\xi})\, w_{\boldsymbol{i}}}{\sum_{\boldsymbol{j}} N_{\boldsymbol{j}}(\boldsymbol{\xi})\, w_{\boldsymbol{j}}}. \tag{2.14}$$

As for B-splines, NURBS geometries are obtained by linear combinations of some coefficients $\boldsymbol{P}_{\boldsymbol{i}}$

$$\boldsymbol{F}(\boldsymbol{\xi}) = \sum_{\boldsymbol{i}} \bar{R}_{\boldsymbol{i}}(\boldsymbol{\xi})\, \boldsymbol{P}_{\boldsymbol{i}}. \tag{2.15}$$

In general, each function $\bar{R}_i(\boldsymbol{\xi})$ cannot be obtained as a tensor-product of univariate functions. However, NURBS geometries $\mathring{\boldsymbol{F}}$ in homogeneous coordinates follow a tensor structure induced by the B-splines

$$\mathring{\boldsymbol{F}}(\boldsymbol{\xi}) = \sum_{\boldsymbol{i}} N_{\boldsymbol{i}}(\boldsymbol{\xi})\, \mathring{\boldsymbol{P}}_{\boldsymbol{i}}. \tag{2.16}$$

2.4 Kronecker product

The tensor-product nature of multivariate B-spline geometries confers a precise structure to linear operators operating on control points or B-spline functions. Given the matrix representations of some linear operators defined in each parametric direction, the tensor product operator is often represented as the *Kronecker product* of these matrices.

Given a matrix $\boldsymbol{Y} \in \mathbb{R}^{y_1 \times y_2}$, and given $\boldsymbol{X} \in \mathbb{R}^{x_1 \times x_2}$, the Kronecker product $\boldsymbol{Y} \otimes \boldsymbol{X}$ is the block matrix

$$\boldsymbol{Y} \otimes \boldsymbol{X} = \begin{bmatrix} Y_{11}\boldsymbol{X} & Y_{12}\boldsymbol{X} & \dots & Y_{1y_2}\boldsymbol{X} \\ Y_{21}\boldsymbol{X} & Y_{22}\boldsymbol{X} & \dots & Y_{2y_2}\boldsymbol{X} \\ \vdots & & & \vdots \\ Y_{y_1 1}\boldsymbol{X} & Y_{y_1 2}\boldsymbol{X} & \dots & Y_{y_1 y_2}\boldsymbol{X} \end{bmatrix} \in \mathbb{R}^{x_1 y_1 \times x_2 y_2}.$$

The block structure of this operation confers various important properties to the Kronecker product. The identities relevant in this work are:

$$\boldsymbol{Z} \otimes (\boldsymbol{Y} \otimes \boldsymbol{X}) = (\boldsymbol{Z} \otimes \boldsymbol{Y}) \otimes \boldsymbol{X} \qquad \text{(associativity)}, \tag{2.17}$$

$$\boldsymbol{Y} \otimes \boldsymbol{X} = (\boldsymbol{Y} \otimes \boldsymbol{I}_{x_1})\, (\boldsymbol{I}_{y_2} \otimes \boldsymbol{X}) \qquad \text{(compatibility)}, \tag{2.18}$$

$$\boldsymbol{Y} \otimes \boldsymbol{X} = \boldsymbol{K}^{(y_1,\, x_1)}\, (\boldsymbol{X} \otimes \boldsymbol{Y})\, \boldsymbol{K}^{(x_2,\, y_2)} \qquad \text{(pseudo-commutativity)}, \tag{2.19}$$

$$(\boldsymbol{A} \otimes \boldsymbol{B})\, (\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{A}\, \boldsymbol{B}) \otimes (\boldsymbol{C}\, \boldsymbol{D}) \qquad \text{(mixed-product)}, \tag{2.20}$$

where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{Z}$ are arbitrary real matrices of compatible dimensions, $\boldsymbol{I}_a \in \mathbb{R}^{a \times a}$ is the identity matrix, and $\boldsymbol{K}^{a,b} \in \mathbb{R}^{ab \times ab}$ is the (perfect shuffle) permutation matrix

$$\begin{aligned} K^{a,b}_{ij} &= \delta_{i,\, (j \ \mathtt{mod}\ a)b + (j \ \mathtt{div}\ a)} \\ &= \delta_{(i \ \mathtt{mod}\ b)a + (i \ \mathtt{div}\ a),\, j}\ , \end{aligned}$$

where δ is the Kronecker delta, while $j \ \mathtt{div}\ a = \lfloor j/a \rfloor$ and $j \ \mathtt{mod}\ a = j - a\lfloor j/a \rfloor$ denote the (positive) integer division and remainder operator, respectively, and $\lfloor x \rfloor$ is the greatest integer less than or equal to x. Further details can be found in, e.g., Fernandes et al. [1998]; Loan [2000] and references therein.

2.5 Isogeometric analysis

In the previous sections, NURBS and B-splines were introduced in their natural geometric context. These concepts are now used to define both the computational domain

(mesh) and the discrete solution space's basis functions. The connection between geometry and analysis is naturally furnished by the isoparametric concept: the same set of basis functions describe both the solution space and the geometry.

This approach is compelling for computational domains that are described by NURBS geometries, as suitable solution spaces can be constructed on the exact geometry. When the domain of interest is not described exactly by NURBS geometries, isogeometric analysis [Cottrell et al., 2009; Hughes et al., 2005] can be combined with the finite cell method [Düster et al., 2008, 2017], retaining the geometric exactness. This approach is discussed in Chapters 6–8. Other techniques such as multi-patch coupling are not considered in this work.

Let $\boldsymbol{F}(\boldsymbol{\xi}) : \hat{\Omega} \to \Omega \subset \mathbb{R}^{m_g}$ be a D-variate NURBS geometry identified by some control points $\boldsymbol{P_i} \in \mathbb{R}^{m_g}$. As in the previous sections, $\boldsymbol{F}(\boldsymbol{\xi})$ is defined as

$$\boldsymbol{F}(\boldsymbol{\xi}) = \sum_{\boldsymbol{i}} \bar{R}_{\boldsymbol{i}}(\boldsymbol{\xi})\, \boldsymbol{P_i}.$$

The functions $\bar{R}_{\boldsymbol{i}}$ are also used as basis functions defining the discrete solution space. In the parametric domain $\hat{\Omega}$, the solution field $\hat{\boldsymbol{u}}^h : \hat{\Omega} \to \mathbb{R}^{m_s}$ is described as

$$\hat{\boldsymbol{u}}^h(\boldsymbol{\xi}) = \sum_{\boldsymbol{i}} \bar{R}_{\boldsymbol{i}}(\boldsymbol{\xi})\, \boldsymbol{d_i},$$

for some degrees of freedom $\boldsymbol{d_i} \in \mathbb{R}^{m_s}$. The solution field in the physical space Ω is obtained as

$$\boldsymbol{u}^h = \hat{\boldsymbol{u}}^h \circ \boldsymbol{F}^{-1}.$$

Note that the solution field $\boldsymbol{u}^h$ has the smoothness induced by the NURBS basis functions $\bar{R}_{\boldsymbol{i}}$.

2.5.1 Refinements

While the computational domain is exactly represented, it might be desirable to enrich the solution space to increase the accuracy of the simulation results. To this end, basis functions suitable for numerical analysis are obtained by "refining" the functions describing the geometry. The refinement can be performed while leaving the geometry and its parametrization unchanged. NURBS functions naturally furnish three main approaches to refinement:

- knot insertion,
- degree elevation, and
- k-refinement.

2.5.1.1 Knot insertion

Given a NURBS curve $\mathring{\boldsymbol{F}} = \sum_{i=1}^{m} N_i \mathring{\boldsymbol{P}}_i$ defined on the knot vector

$$\boldsymbol{\Xi} = (\xi_1, ..., \xi_{m+p+1}),$$

the objective is to describe the same curve in a different basis determined by a knot vector

$$\tilde{\boldsymbol{\Xi}} = (\tilde{\xi}_1, ..., \tilde{\xi}_{\tilde{m}+p+1})$$

obtained by adding knots to $\boldsymbol{\Xi}$. Namely, each knot ξ_i in $\boldsymbol{\Xi}$ is also present in $\tilde{\boldsymbol{\Xi}}$. This inclusion is indicated by $\boldsymbol{\Xi} \subset \tilde{\boldsymbol{\Xi}}$. Note that the knot multiplicities in $\tilde{\boldsymbol{\Xi}}$ are at least the multiplicities in $\boldsymbol{\Xi}$.

Let $\{\tilde{N}_i\}$ be the B-splines functions defined on the knot vector $\tilde{\boldsymbol{\Xi}}$. Since $\boldsymbol{\Xi} \subset \tilde{\boldsymbol{\Xi}}$, the same inclusion holds in the respective B-spline spaces:

$$span\{N_i\} \subset span\left\{\tilde{N}_i\right\}, \tag{2.21}$$

as both are piecewise-polynomial spaces of the same order, and each $\tilde{N}_i$ has a number of continuous derivatives at the knots that is less or equal than the same number for N_i. The function spaces' inclusion in Equation (2.21) implies that the curve $\mathring{\boldsymbol{F}} = \sum_{i=1}^{m} N_i \mathring{\boldsymbol{P}}_i$ admits the representation

$$\mathring{\boldsymbol{F}}(\boldsymbol{\xi}) = \sum_{i=1}^{\tilde{m}} \tilde{N}_i \, \mathring{\boldsymbol{Q}}_i, \tag{2.22}$$

for some control points $\{\mathring{\boldsymbol{Q}}_i\}$. The representations in terms of the functions $\{\tilde{N}_i\}$ or $\{N_i\}$ are expressions of the same curve, maintaining the geometry and its parametrization unchanged. The knot-insertion procedure determines the coefficients $\{\mathring{\boldsymbol{Q}}_i\}$ associated with this change of basis.

If only a single knot $\bar{\xi} \in [\xi_k, \xi_{k+1})$ is inserted at the $(k+1)$th position, i.e.,

$$\tilde{\boldsymbol{\Xi}} = (\tilde{\xi}_1 = \xi_1, ..., \tilde{\xi}_k = \xi_k, \tilde{\xi}_{k+1} = \bar{\xi}, \tilde{\xi}_{k+2} = \xi_{k+1}, ..., \tilde{\xi}_{m+p+2} = \xi_{m+p+1}),$$

an explicit formula for the coefficients $\{\mathring{\boldsymbol{Q}}_i\}$ can be obtained from Equation (2.3). Namely,

$$\mathring{\boldsymbol{Q}}_i = \alpha_i \mathring{\boldsymbol{P}}_i + (1 - \alpha_i) \mathring{\boldsymbol{P}}_{i-1}, \tag{2.23}$$

where

$$\alpha_i = \begin{cases} 1 & \text{if } i \leq k - p, \\ \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & \text{if } k - p < i \leq k, \\ 0 & \text{if } i > k. \end{cases} \tag{2.24}$$

When more than one knot is added to $\boldsymbol{\Xi}$, this procedure can be iterated to insert an arbitrary number of knots, one at a time.

Equation (2.23) shows that the knot insertion is a linear operation. Therefore, it can be expressed in matrix form as

$$\mathring{\boldsymbol{Q}} = \boldsymbol{R}^{\top} \mathring{\boldsymbol{P}}, \tag{2.25}$$

where

$$\mathring{Q}_{ij} = \left[\mathring{\boldsymbol{Q}}_i\right]_j, \tag{2.26}$$

$$\mathring{P}_{ij} = \left[\mathring{\boldsymbol{P}}_i\right]_j. \tag{2.27}$$

The operator $\boldsymbol{R}$ is referred to as the *knot-insertion operator*.

In D-variate spaces defined by the knot vectors

$${}^{d}\boldsymbol{\Xi} = \left({}^{d}\xi_1, ..., {}^{d}\xi_{{}^{d}m+{}^{d}p+1}\right), \qquad d = 1, ..., D, \tag{2.28}$$

knots can be inserted in the knot vector ${}^{d}\boldsymbol{\Xi}$ for each parametric direction d, yielding the univariate knot-insertion operators $\{{}^{d}\boldsymbol{R}\}_{d=1,...,D}$. In order to use a unified matrix notation, the refined control points $\{\mathring{\boldsymbol{Q}}_{\boldsymbol{i}}\}$, with multi-index $\boldsymbol{i} = (i_1, ..., i_D)$, can be arranged in a matrix $\mathring{\boldsymbol{Q}}$. Specifically, the multi-index $\boldsymbol{i}$ can be identified by the scalar index $i = i(\boldsymbol{i})$ following the canonical tensor-product order

$$i = i_1 + {}^{1}m(i_2 + {}^{2}m(...i_{D-1} + {}^{D-1}m\ i_D)). \tag{2.29}$$

The matrix $\mathring{\boldsymbol{Q}}$ is defined as

$$\mathring{Q}_{ij} = \mathring{Q}_{i(\boldsymbol{i}),j} = \left[\mathring{\boldsymbol{Q}}_{\boldsymbol{i}}\right]_j. \tag{2.30}$$

The knot-insertion operator for the tensor-product geometry is the tensor product of the univariate operators

$$\mathring{\boldsymbol{Q}} = \left({}^{D}\boldsymbol{R} \otimes ... \otimes {}^{1}\boldsymbol{R}\right)^{\top} \mathring{\boldsymbol{P}}. \tag{2.31}$$

2.5.1.2 Degree elevation

The degree elevation is a procedure similar to knot insertion in the sense that a given curve is expressed in the coordinates of a finer space. In the case of knot insertion, the finer space consists of "more functions," while in the case of degree elevation, it consists of higher-order functions.

Specifically, given an open-knot vector

$$\boldsymbol{\Xi}^{p} = (\underbrace{t_1, ..., t_1}_{p+1}, \underbrace{t_2, ..., t_2}_{r_2}, ..., \underbrace{t_{s-1}, ..., t_{s-1}}_{r_{s-1}}, \underbrace{t_s, ..., t_s}_{p+1}),$$

with associated pth-degree basis functions $\boldsymbol{N}^p$, a function space including $span\{N_i^p\}$ is generated by the knot vector $\boldsymbol{\Xi}^p$ obtained by raising all multiplicities by one, that is

$$\boldsymbol{\Xi}^{p+1} = (\underbrace{t_1, ..., t_1}_{p+2}, \underbrace{t_2, ..., t_2}_{r_2+1}, ..., \underbrace{t_{s-1}, ..., t_{s-1}}_{r_{s-1}+1}, \underbrace{t_s, ..., t_s}_{p+2}).$$

The B-splines defined by $\boldsymbol{\Xi}^{p+1}$ possess the same smoothness as the functions $\boldsymbol{N}^p$ but are of degree $p+1$. Given a NURBS curve $\mathring{\boldsymbol{F}} = (\boldsymbol{N}^p)^\top \mathring{\boldsymbol{P}}^p$, the nestedness condition

$$span\{N_i^p\} \subset span\left\{N_i^{p+1}\right\} \tag{2.32}$$

implies the existence of some control points $\mathring{\boldsymbol{P}}^{p+1}$, such that

$$(\boldsymbol{N}^p)^\top \mathring{\boldsymbol{P}}^p = \mathring{\boldsymbol{F}} = (\boldsymbol{N}^{p+1})^\top \mathring{\boldsymbol{P}}^{p+1}. \tag{2.33}$$

Explicit algorithms to determine the coefficients $\mathring{\boldsymbol{P}}^{p+1}$ from $\mathring{\boldsymbol{P}}^p$ are given in, e.g., Piegl and Tiller [1995].

2.5.1.3 k-refinement

Knot insertion and degree elevation do not commute. For example, consider the knot vector

$$\boldsymbol{\Xi} = (-1, -1, 1, 1).$$

Inserting one knot $\xi = 0$ and, afterwards, elevating the degree, yields

$$\tilde{\boldsymbol{\Xi}} = (-1, -1, -1, 0, 0, 1, 1, 1).$$

Instead, performing the knot insertion after the degree elevation gives

$$\tilde{\boldsymbol{\Xi}} = (-1, -1, -1, 0, 1, 1, 1).$$

The first knot-vector generates functions that are $\mathcal{C}^0$ at $\xi = 0$, while the second knot-vector generates functions that are $\mathcal{C}^1$ at the same location.

The k-refinement is the procedure obtained by first elevating the degree and then inserting new knots. This approach produces functions that have the maximum smoothness dictated by the underlying geometry.

2.6 Standard Bézier extraction

The isogeometric approach presented in Section 2.5 shows a few differences compared to standard finite element implementation. Considering every non-degenerate knot span as a finite element, one crucial difference is that the shape functions are not the same for each element. In particular, a knot vector can contain different multiplicities, defining different element basis functions while keeping the element boundaries unchanged. In this

sense, it is not possible to adopt the standard implementation strategy based on defining a set of standard basis functions on a reference space shared by all finite elements. This difference makes it harder to integrate isogeometric analysis into an already-existing finite element code. The Bézier extraction is proposed in Borden et al. [2011] to mitigate these difficulties and, in this section, its main ideas and properties are reviewed, following closely Borden et al. [2011]; Piegl and Tiller [1995].

Consider a univariate knot vector

$$\boldsymbol{\Xi} = (\xi_1, ..., \xi_{m+p+1})$$

with an associated vector $\boldsymbol{N}$ composed of pth-degree B-spline basis functions. Let

$$\mathring{\boldsymbol{F}}(\xi) = \mathring{\boldsymbol{P}}^\top \boldsymbol{N}(\xi)$$

be a NURBS geometry defined by the homogeneous control points $\mathring{\boldsymbol{P}} \in \mathbb{R}^{m\times(m_g+1)}$, m_g being the number of components of each control point. The Bézier decomposition of $\mathring{\boldsymbol{F}}$ into piecewise Bézier curves is obtained by raising to p the multiplicity of each internal knot in $\boldsymbol{\Xi}$. This operation can be done by inserting new knots in $\boldsymbol{\Xi}$, as explained in Section 2.5.1.1. This way, the same spline $\mathring{\boldsymbol{F}}$ can be represented in terms of the Bernstein polynomials $\boldsymbol{B}$, as

$$\mathring{\boldsymbol{F}}(\xi) = \mathring{\boldsymbol{Q}}^\top \boldsymbol{B}(\xi).$$

Increasing the knot repetition is a linear operation, i.e., the Bernstein control points $\mathring{\boldsymbol{Q}}$ can be obtained as a linear combination of the B-spline control points $\mathring{\boldsymbol{P}}$. The matrix $\boldsymbol{E}$ representing this operation is called the Bézier extraction operator. In particular, $\boldsymbol{E}$ is such that

$$\mathring{\boldsymbol{Q}} = \boldsymbol{E}^\top \mathring{\boldsymbol{P}}. \tag{2.34}$$

This operator can also be used to obtain the dual relation, i.e., to express the B-spline functions $\boldsymbol{N}$ as a linear combination of Bernstein polynomials $\boldsymbol{B}$. Indeed,

$$\begin{aligned}
\mathring{\boldsymbol{P}}^\top \boldsymbol{N} &= \mathring{\boldsymbol{F}} \\
&= \mathring{\boldsymbol{Q}}^\top \boldsymbol{B} \\
&= \left(\mathring{\boldsymbol{P}}^\top \boldsymbol{E}\right) \boldsymbol{B} && \text{(Equation (2.34))} \\
&= \mathring{\boldsymbol{P}}^\top (\boldsymbol{E}\boldsymbol{B}) .
\end{aligned}$$

The arbitrarity of $\mathring{\boldsymbol{P}}$ yields the dual relation

$$\boldsymbol{N} = \boldsymbol{E}\boldsymbol{B}. \tag{2.35}$$

2.6.1 Element-local Bézier extraction

Relations (2.34) and (2.35) can be restricted to a single element $\tilde{\Omega}^e$ in the knot-vector parameter space. Namely, $\tilde{\Omega}^e$ is defined by a non-degenerate knot span $\tilde{\Omega}^e = [\xi_1^e, \xi_2^e]$, for some distinct knot values $\xi_1^e < \xi_2^e$ in $\boldsymbol{\Xi}$. In particular, the element-local relations read

$$\mathring{\boldsymbol{Q}}^e = (\boldsymbol{E}^e)^\top \mathring{\boldsymbol{P}}^e, \tag{2.36}$$

$$\boldsymbol{N}^e = \boldsymbol{E}^e \boldsymbol{B}^e, \tag{2.37}$$

where $\boldsymbol{N}^e$ and $\boldsymbol{B}^e$ are column vectors consisting of $\boldsymbol{N}$ and $\boldsymbol{B}$ functions having support on element $\tilde{\Omega}^e$, respectively. $\mathring{\boldsymbol{P}}^e$ and $\mathring{\boldsymbol{Q}}^e$ are the control points associated with the functions $\boldsymbol{N}^e$, $\boldsymbol{B}^e$, respectively. $\boldsymbol{E}^e$ is obtained from $\boldsymbol{E}$ by selecting the rows corresponding to functions in $\boldsymbol{N}^e$ and columns associated with functions in $\boldsymbol{B}^e$. The localization of operators is further discussed in Chapter 4. Throughout this dissertation, the superscript e indicates element-local matrices.

The element functions $\boldsymbol{N}^e(\xi)$ and $\boldsymbol{B}^e(\xi)$ are defined for any $\xi \in \tilde{\Omega}^e$. The functions $\boldsymbol{B}^e(\xi)$ are the same for each element $\tilde{\Omega}^e$ up to a coordinate change. In particular, it is possible to obtain $\boldsymbol{B}^e(\xi)$ from the Bernstein polynomials $\boldsymbol{B}(\hat{\xi})$ defined on a reference interval $\hat{\Omega} = [a, b]$, i.e.,

$$B_i(\hat{\xi}) = \binom{p}{i-1} \frac{(b-\hat{\xi})^{p-(i-1)}(\hat{\xi}-a)^{i-1}}{(b-a)^p}, \qquad i = 1, ..., p+1,$$

where

$$\binom{p}{i-1} = \frac{p!}{p!(i-1)!}.$$

Without loss of generality, it is assumed $\hat{\Omega} = [-1, 1]$ to facilitate Gaussian quadrature.

Given the mapping from reference space to the parametric element $\tilde{F}^e : \hat{\Omega} \to \tilde{\Omega}^e$, such that $\tilde{F}^e(\hat{\Omega}) = \tilde{\Omega}^e$, it holds

$$\boldsymbol{B}^e\left(\tilde{F}^e(\hat{\xi})\right) = \boldsymbol{B}(\hat{\xi}).$$

Therefore, the element basis functions $\boldsymbol{N}^e(\tilde{F}^e(\hat{\xi})) = \boldsymbol{N}^e(\hat{\xi})$ defined on the reference space $\hat{\Omega}$ can be expressed through Bernstein polynomials $\boldsymbol{B}(\hat{\xi})$ as

$$\boldsymbol{N}^e(\hat{\xi}) = \boldsymbol{E}^e \boldsymbol{B}(\hat{\xi}). \tag{2.38}$$

Equation (2.38) defines the element shape functions $\boldsymbol{N}^e$ from a set of element-independent Bernstein polynomials $\boldsymbol{B}$ defined on a common reference space $\hat{\Omega}$.

Equation (2.36) determines a set of element control points defining a direct mapping $F^e : \hat{\Omega} \to \Omega^e$ from the reference space $\hat{\Omega}$ to the physical space Ω^e. Specifically,

$$F^e(\hat{\xi}) = (\mathring{\boldsymbol{Q}}^e)^\top \boldsymbol{B}(\hat{\xi}).$$

In summary, the Bézier extraction operator defines a set of basis functions $\boldsymbol{B}$ on a reference space $\hat{\Omega}$ common to each element. It also determines the control points to obtain a direct geometric mapping to the (unchanged) geometry of the problem through the same functions $\boldsymbol{B}$. This operator furnishes a strategy that is one step closer to traditional finite element implementations based on a common reference space. The only additional step required consists of applying the transformation $\boldsymbol{E}^e$ given in Equation (2.38). This modification can be confined to shape-function evaluation subroutines.

2.6.2 Multivariate Bézier extraction

In D-variate domains, $\boldsymbol{E}$, $\boldsymbol{N}$, and $\boldsymbol{B}$ are a tensor product of D univariate operators obtained as described in the previous section. For example, for $D = 2$, it holds

$$\begin{aligned}\boldsymbol{N} &= {}^2\boldsymbol{N} \otimes {}^1\boldsymbol{N},\\ \boldsymbol{B} &= {}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B},\end{aligned}$$

where the left superscript indicates the parametric direction. Using Equation (2.35) in each direction yields

$$\begin{aligned}{}^2\boldsymbol{N} \otimes {}^1\boldsymbol{N} &= \left({}^2\boldsymbol{E}\,{}^2\boldsymbol{B}\right) \otimes \left({}^1\boldsymbol{E}\,{}^1\boldsymbol{B}\right), &(2.39)\\ &= \left({}^2\boldsymbol{E} \otimes {}^1\boldsymbol{E}\right)\left({}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B}\right). &(2.40)\end{aligned}$$

Namely, the fundamental relations Equations (2.35) and (2.37) hold also for tensor product functions $\boldsymbol{N}$ and $\boldsymbol{B}$, in which the extraction operator $\boldsymbol{E}$ is the tensor product of the univariate extraction operators. Also Equations (2.34) and (2.36) hold for control points of compatible size. Using the same notation as in [Thomas et al., 2015], such a reverse Kronecker product is denoted by

$$\boldsymbol{E} = \bigodot_{i=1}^{D} {}^i\boldsymbol{E} = {}^D\boldsymbol{E} \otimes {}^{D-1}\boldsymbol{E} \otimes \ldots \otimes {}^2\boldsymbol{E} \otimes {}^1\boldsymbol{E}.$$

For example, in three dimensions, the above expression corresponds to

$$\boldsymbol{E} = \bigodot_{i=1}^{3} {}^i\boldsymbol{E} = {}^3\boldsymbol{E} \otimes {}^2\boldsymbol{E} \otimes {}^1\boldsymbol{E}.$$

Chapter 3

Hierarchical B-splines

The previous chapter introduced isogeometric analysis, illustrating the tensor-product structure of both the geometry and the solution field. This property can become a disadvantage when approximating fields featuring localized phenomena, such as singularities or stress concentrations. For example, inserting a knot in one parametric direction produces not only new functions with support in the proximity of desired areas but also new functions along the whole extent of the remaining parametric directions. This effect is undesirable, as it precludes local refinement.

This chapter reviews one approach to local refinement based on the definition of B-splines on multiple levels with increasingly smaller support. The final set of basis functions describing the discrete solution space comprises functions belonging to different levels, allowing to introduce refined functions only where needed. This technique, named hierarchical B-splines ($\mathcal{HB}$), is presented in, e.g., Forsey and Bartels [1988]; Greiner and Hormann [1997]; Kraft [1997]; Vuong et al. [2011]. The current chapter completes the preliminaries for the new contributions elaborated in the following chapters.

3.1 Introduction

Adaptivity has become a fundamental topic of research in isogeometric analysis. Different techniques are being currently developed, including T-Splines (see, e.g., Bazilevs et al. [2010]; Beirão da Veiga et al. [2012]; Beirão da Veiga et al. [2013]; Li and Scott [2012]; Li et al. [2012]; Scott et al. [2012, 2011]), LR-Splines (see, e.g., Dokken et al. [2013]), hierarchical T-splines (see, e.g., Evans et al. [2015]), and hierarchical B-splines ($\mathcal{HB}$) (see, e.g., Forsey and Bartels [1988]; Greiner and Hormann [1997]; Kraft [1997]; Vuong et al. [2011]). This strategy has been recently improved under the name of truncated hierarchical B-splines ($\mathcal{THB}$) [Giannelli et al., 2016, 2012], aiming to recover the partition of unity property and enhance the bandwidth granted by standard hierarchical B-splines.

The hierarchical B-splines and their truncated variant are based on a hierarchical definition of functions subdivided into multiple levels, as described in the following sections following closely Giannelli et al. [2016]; Hennig et al. [2016].

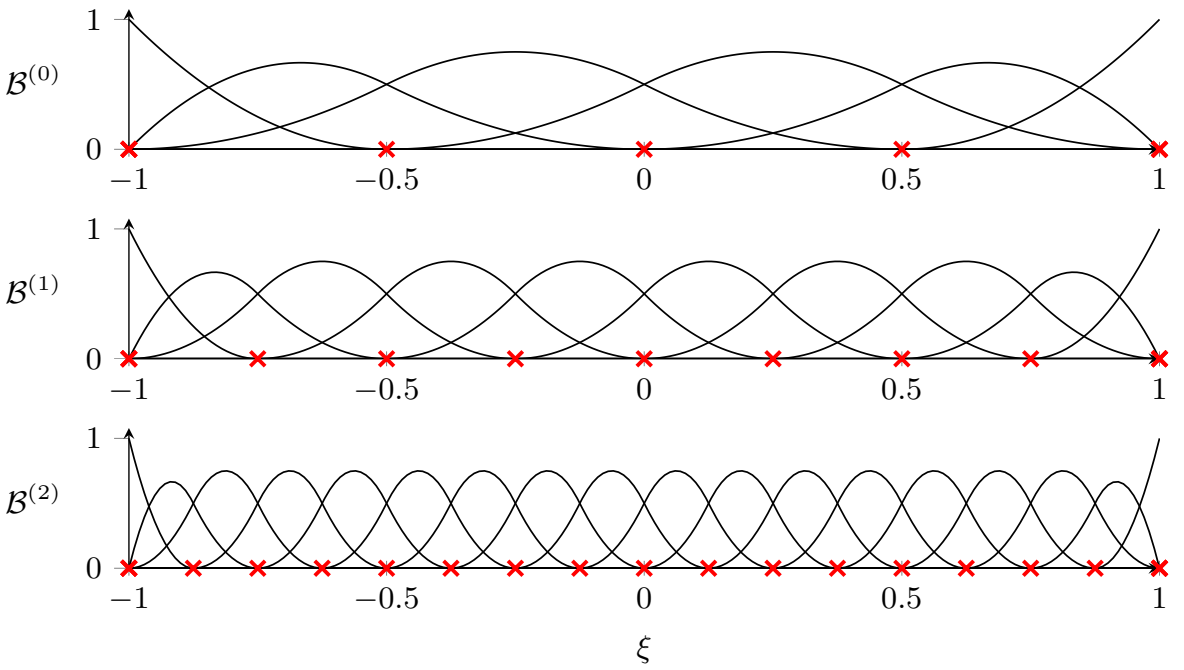


Figure 3.1: Example of hierarchical B-spline functions. The knots of each level are marked by red crosses.

3.2 Function hierarchy

Let

$$\mathcal{V}^{(0)} \subset \mathcal{V}^{(1)} \subset \cdots \subset \mathcal{V}^{(L)} \tag{3.1}$$

be a sequence of nested B-spline spaces defined on a D-variate parametric domain $\Omega \subset \mathbb{R}^D$. Each space $\mathcal{V}^{(l)}$ is assumed to have a finite dimension denoted by $dim(\mathcal{V}^{(l)})$. It is spanned by the pth-degree B-spline basis, $\mathcal{B}^{(l)}$, identified by the knot vectors

$${}^d\Xi^{(l)} = ({}^d\xi_1^{(l)}, \ldots, {}^d\xi_{{}^dm^{(l)}+p+1}^{(l)}), \qquad 1 \leq d \leq D,\ 0 \leq l \leq L.$$

Note that the degree p is the same for all parametric directions. The nested nature of the spaces $\mathcal{V}^{(l)}$ implies that the knot vectors are also nested, i.e., ${}^d\Xi^{(l)} \subset {}^d\Xi^{(l+1)}$ (see Section 2.5.1.1).

Figure 3.1 shows an example consisting of three nested knot vectors together with the associated B-spline functions. In particular, note that each level-l knot value is also present in every knot vector of a higher level. The base level ($l = 0$) is referred to as coarse level, while the others are called finer levels.

Among these B-splines defined on different levels, it is of interest to identify a suitable set of functions

$$\mathcal{H} \subset \bigcup_{l=0}^{L} \mathcal{B}^{(l)}$$

used as basis functions for both analysis and geometry description. The choice for $\mathcal{H}$ is driven by some areas of interest, where finer functions should be activated to selectively increase the resolution of the space.

To this end, Ω is partitioned into a finite subdomain set $\mathcal{T} = \{\Omega^e\}_{e=1,\dots,n_e}$ composed of nonempty knot spans of any level. Specifically, given the set $\mathcal{Q}^{(l)}$ of nonempty knot spans defined by the level-l knot vectors $\{{}^d\boldsymbol{\Xi}^{(l)}\}_{d=1,\dots,D}$, the multi-level mesh can be identified by any set

$$\mathcal{T} \subset \bigcup_l \mathcal{Q}^{(l)}, \tag{3.2}$$

such that

$$\Omega = \bigcup_{\Omega^e \in \mathcal{T}} \overline{\Omega^e}, \tag{3.3}$$

where $\overline{\Omega^e}$ denotes the closure of Ω^e. Figure 3.2 shows a possible choice for $\mathcal{T}$ in the knot-span hierarchy considered in Figure 3.1.

The set of level-l elements is denoted by

$$\mathcal{T}^{(l)} = \mathcal{T} \cap \mathcal{Q}^{(l)}, \tag{3.4}$$

and let $\Omega^{(l)}$ be their domain, i.e.,

$$\Omega^{(l)} = \bigcup_{\Omega^e \in \mathcal{T}^{(l)}} \overline{\Omega^e}. \tag{3.5}$$

To identify a suitable choice of hierarchical basis functions, let

$$\Omega_+^{(l)} = \bigcup_{l^*=l+1}^{L} \Omega^{(l^*)},$$
$$\Omega_-^{(l)} = \bigcup_{l^*=0}^{l-1} \Omega^{(l^*)},$$

be respectively the finer and coarser domains with respect to the lth level. In the example given in Figure 3.2, these domains consist of the following intervals

$$\begin{array}{lll} \Omega^{(0)} = [-1,0], & \Omega^{(1)} = [0,0.25], & \Omega^{(2)} = [0.25,1], \\ \Omega_+^{(0)} = [0,1], & \Omega_+^{(1)} = [0.25,1], & \Omega_+^{(2)} = \emptyset, \\ \Omega_-^{(0)} = \emptyset, & \Omega_-^{(1)} = [-1,0], & \Omega_-^{(2)} = [-1,0.25]. \end{array}$$

Based on the partition $\mathcal{T}$, the basis functions $\mathcal{H}$ can be defined through the set $\mathcal{B}_a^{(l)}$ of functions with support on some elements, i.e.,

$$\mathcal{B}_a^{(l)} = \left\{ N \in \mathcal{B}^{(l)} \ : \ supp(N) \cap \Omega^{(l)} \neq \emptyset \right\} \subset \mathcal{B}^{(l)}.$$

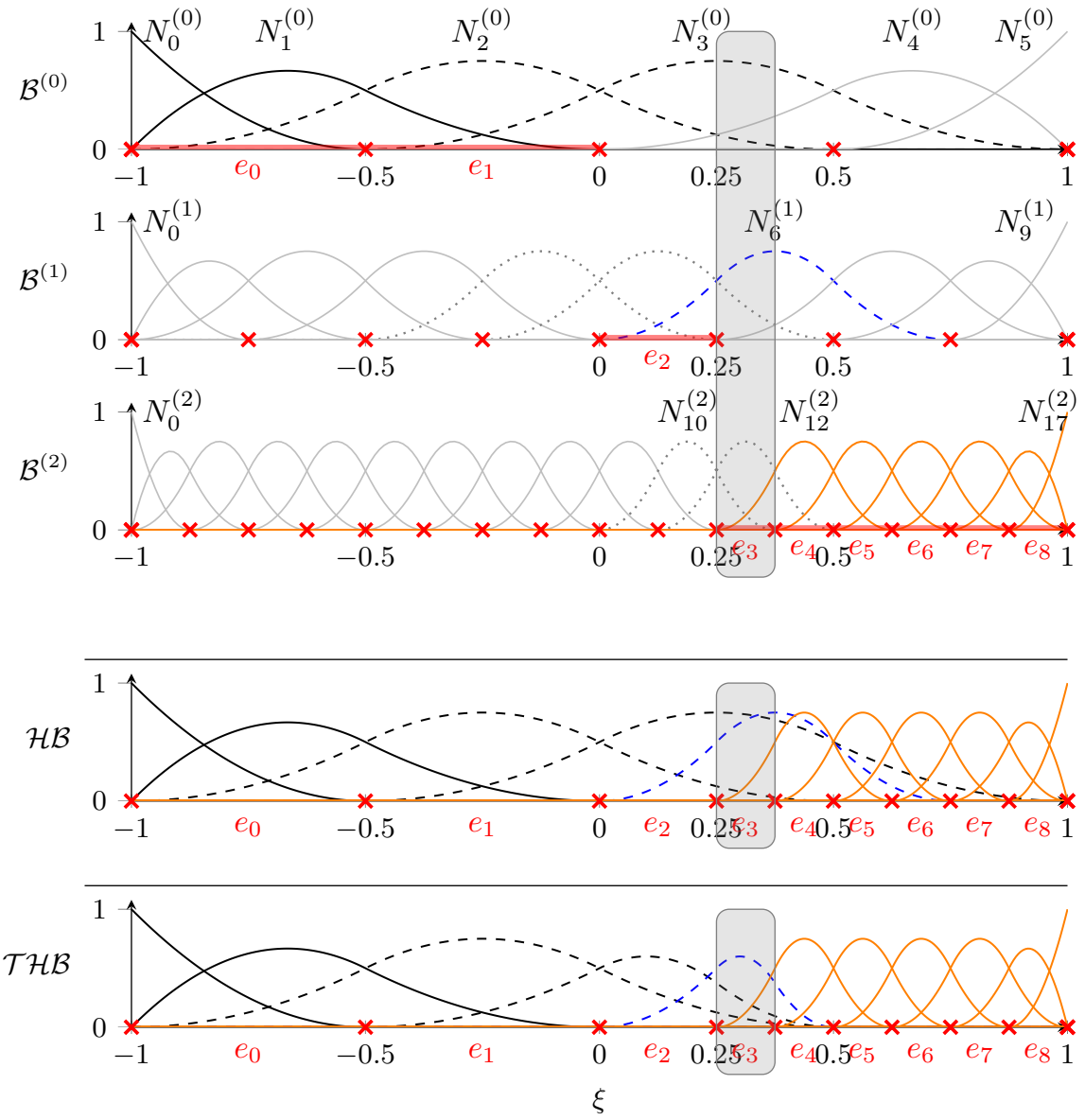


Figure 3.2: Example of element mesh and (truncated) hierarchical B-spline basis. The elements are marked by the thick red lines on the horizontal axes. $\mathcal{B}_a^{(l)}$ is composed of the colored or dotted functions. $\mathcal{B}_-^{(l)}$ is composed of the dotted functions. $\mathcal{B}_+^{(l)}$ is composed of the dashed functions. $\mathcal{B}_=^{(l)}$ is composed of the solid non-gray functions. The solid gray functions are inactive.

See Figure 3.2 for an example. To identify a subset of linearly independent functions, $\mathcal{B}_a^{(l)}$ is divided into the sets

$$\mathcal{B}_-^{(l)} = \left\{ N \in \mathcal{B}_a^{(l)} : \mathit{supp}(N) \cap \Omega_-^{(l)} \neq \emptyset \right\} \quad \textit{(overlapping coarser elements)},$$

$$\mathcal{B}_+^{(l)} = \left\{ N \in \mathcal{B}_a^{(l)} : \mathit{supp}(N) \cap \Omega_+^{(l)} \neq \emptyset \right\} \setminus \mathcal{B}_-^{(l)} \quad \textit{(overlapping finer elements)},$$

$$\mathcal{B}_=^{(l)} = \left\{ N \in \mathcal{B}_a^{(l)} : \mathit{supp}(N) \subset \Omega^{(l)} \right\} \quad \textit{(overlapping same-level elements)}.$$

These subsets are illustrated in Figure 3.2. The above definitions are used in the following sections to define the hierarchical B-spline basis and its truncated variant.

3.3 Hierarchical B-splines

The hierarchical B-splines basis [Kraft, 1997; Vuong et al., 2011] $\mathcal{H} = \mathcal{HB}$ is defined as

$$\mathcal{HB} = \bigcup_{l=0}^{L} \mathcal{HB}^{(l)}, \tag{3.6}$$

where

$$\mathcal{HB}^{(l)} = \left(\mathcal{B}_{=}^{(l)} \cup \mathcal{B}_{+}^{(l)} \right).$$

Namely, $\mathcal{HB}$ is the set of B-splines of each level l whose support covers only elements of level $l^* \geq l$ and at least one level-l element (see Figure 3.2). It was proven in Vuong et al. [2011] that the hierarchical B-splines are linearly independent.

3.4 Truncated hierarchical B-splines

The truncated hierarchical B-spline basis ($\mathcal{THB}$) [Giannelli et al., 2012] is similar to the hierarchical B-splines, with the main difference being that the functions whose support covers finer elements are shrunk. This approach generates a basis that spans the same space as the hierarchical B-splines [Giannelli et al., 2012] but is composed of functions that

- have smaller support,
- form a partition of unity, and
- have superior stability properties.

The above properties are all desirable in the context of numerical simulations. See Giannelli et al. [2012, 2013] for further details.

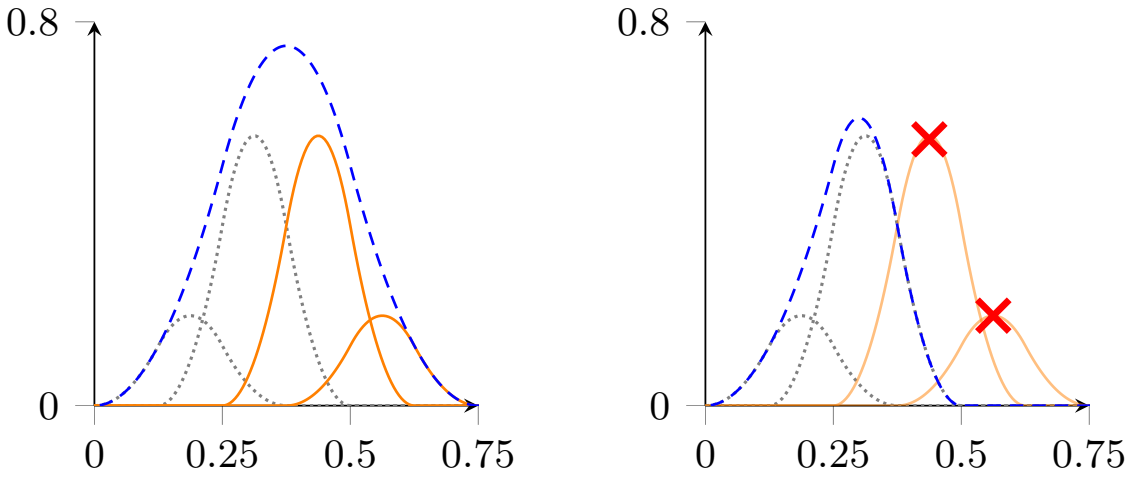
The truncation concept is based on the following definition.

Definition 3.4.1: Truncation operator [Giannelli et al., 2012]
Let $\tau \in \mathcal{V}^{(l)}$, and let

$$\tau = \sum_{N \in \mathcal{B}^{(l+1)}} c_N^{(l+1)}(\tau)\ N, \qquad\qquad c_N^{(l+1)} \in \mathbb{R}\,, \tag{3.7}$$

be its representation with respect to the finer basis $\mathcal{B}^{(l+1)}$ of $\mathcal{V}^{(l+1)}$. The truncation of τ with respect to the level $l+1$ is defined as

$$\operatorname{trunc}^{(l+1)}(\tau) = \sum_{\substack{N \in \mathcal{B}^{(l+1)}, \\ \operatorname{supp}(N) \cap \Omega_{-}^{(l+1)} \neq \emptyset}} c_N^{(l+1)}(\tau)\, N.$$



(a) $N_6^{(1)} = \frac{1}{4}N_{10}^{(2)} + \frac{2}{4}N_{11}^{(2)} + \frac{3}{4}N_{12}^{(2)} + \frac{1}{4}N_{13}^{(2)}$. (b) $trunc^{(2)}(N_6^{(1)}) = \frac{1}{4}N_{10}^{(2)} + \frac{3}{4}N_{11}^{(2)} + 0N_{12}^{(2)} + 0N_{13}^{(2)}$.

Figure 3.3: The function $N_6^{(1)}$ of the example in Figure 3.2 (left, dashed blue) and its truncation (right, dashed blue) expressed in terms of the finer functions (from left to right): $N_{10}^{(2)}$, $N_{11}^{(2)}$, $N_{12}^{(2)}$, and $N_{13}^{(2)}$ (dotted gray and solid orange).

Namely, the truncation is based on the finer-basis representation, keeping only the contribution of basis functions whose support overlaps coarser elements. Figure 3.3 shows a graphical representation of this procedure. Note that the finer-level representation in Equation (3.7) is always well-defined, as $\mathcal{V}^{(l)} \subset \mathcal{V}^{(l+1)}$.

The truncated hierarchical B-spline basis is obtained by a recursive application of the truncation defined in Definition 3.4.1, as described in the following definition.

Definition 3.4.2: Recursive truncation operator [Buffa and Giannelli, 2016]
Let $\tau \in \mathcal{V}^{(l)}$. The recursive truncation of τ is defined as

$$\mathrm{Trunc}^{(l+1)}(\tau) = \mathrm{trunc}^{(L)}\left(\mathrm{trunc}^{(L-1)}\left(\ldots\ \mathrm{trunc}^{(l+1)}(\tau)\ \ldots\right)\right).$$

Similarly, the recursive truncation of the set $\mathcal{B}^{(l)}$ is the set of truncated functions

$$\mathrm{Trunc}^{(l+1)}\left(\mathcal{B}^{(l)}\right) = \left\{\mathrm{Trunc}^{(l+1)}(N) : N \in \mathcal{B}^{(l)}\right\}.$$

The truncated hierarchical B-splines' definition is similar to the one of hierarchical B-splines, with the difference being that active functions with support partly on finer elements are (recursively) truncated

$$\mathcal{THB} = \bigcup_l \mathcal{THB}^{(l)}, \tag{3.8}$$

where

$$\begin{aligned}\mathcal{THB}^{(l)} &= \mathcal{B}_{=}^{(l)} \cup\ Trunc^{(l+1)}\left(\mathcal{B}_{+}^{(l)}\right)\\ &= Trunc^{(l+1)}\left(\mathcal{HB}^{(l)}\right).\end{aligned}$$

Figure 3.2 shows an example of a truncated hierarchical B-spline basis compared to the standard hierarchical B-splines. Note that the truncated functions spanning multiple levels have smaller support compared to their un-truncated counterpart.

Chapter 4

Multi-level Bézier extraction

Chapter 2 introduced the Bézier extraction for isogeometric analysis (without local refinement) as a way to ease its implementation in already-existing software. In Chapter 3, it was noted that the tensor-product structure of NURBS patches precludes local refinement. The hierarchical B-splines were presented as a possible way to overcome this difficulty.

In this chapter, the concept of Bézier extraction is generalized to any hierarchical refinement based on a sequence of nested function spaces. The proposed strategy shows properties analogous to the standard Bézier extraction, bringing the local refinement one step closer to traditional finite element implementations. Particular focus is devoted to hierarchical B-splines and their truncated variant. However, the proposed strategy is not limited to these approaches, and further generalizations are briefly discussed.

4.1 Introduction

In the literature, several works moving along similar research lines can be found.

In Garau and Vázquez [2016], a similar approach is applied globally following a level-wise strategy, where the active entries of the system matrices are computed in a standard way for each level. Each level's system matrix is successively transformed to produce the entries in the hierarchical B-spline basis. One advantage of this approach is that no connectivity information is needed.

Another similar global approach can be found in Hennig et al. [2016], where the active entries of the system matrices are computed with the standard basis of each hierarchical level. Consecutively, the obtained matrices are inserted into a diagonal block-matrix that is ultimately transformed into the hierarchical system. In Hennig et al. [2016]; Vuong [2014], the Bézier extraction is applied level-wise in a standard fashion.

A very similar concept is outlined in Evans et al. [2015]; Lorenzo et al. [2017]; Scott et al. [2014], where the hierarchical functions are constructed from the Bernstein polynomials. Here, the truncation is not considered, and the presentation is specific to the considered basis functions.

A similar strategy is presented in Apprich et al. [2014]; Bornemann and Cirak [2013], but it is restricted to the overlay of uniform knot-vectors obtained by bisection. In particular, it is not permitted to have knot repetitions, precluding the use of open-knot vectors or reduced-continuity bases. Moreover, the approach does not extend to truncated hierarchical B-splines. Additionally, the method of Bornemann and Cirak [2013] presents the same restrictions as in Kraft [1997], i.e., refinements on the boundary are not allowed.

This dissertation proposes an element-local approach aiming at easing the introduction of local refinement into already-existing software. This approach preserves the "element" concept used in standard finite-element implementations. The proposed approach retains the same properties making the traditional Bézier extraction an effective strategy. The method is defined in a general setting, separating the multi-level extraction from the standard Bézier extraction, setting a more general independent framework applicable to any sequence of nested spaces. The method can be combined with the traditional Bézier extraction, obtaining an extraction for the (truncated) hierarchical B-splines. There is no restriction on the knot vectors' multiplicities, allowing open-knot vectors or reducing the continuity in the finer levels. The method can also be combined with other kinds of standard polynomial functions, such as Lagrange polynomials, integrated Legendre polynomials, or the standard B-splines.

4.2 Basic idea

Consider the hierarchical B-spline basis depicted in Figure 3.2 and the level-two element

$$\Omega^{e_3} = (0.25, 0.375)$$

marked by the gray overlay box. The hierarchical B-splines restricted to Ω^{e_3} can be obtained by a linear combination of the single-level functions in $\mathcal{B}^{(2)}$ with support on Ω^{e_3}. Namely, the element-local hierarchical basis of Ω^{e_3} is a linear combination of the standard B-Splines defined by the level-two knot vector $\boldsymbol{\Xi}^{(2)}$. This concept is illustrated in Figure 4.1.

This representation is valid for each level-l element Ω^e: the element-local hierarchical basis can be represented on Ω^e as a linear combination of standard level-l B-spline functions in $\mathcal{B}^{(l)}$. This concept will be further explained later in the section. Note that the B-spline basis $\mathcal{B}^{(l)}$ belongs to the same level l as the element Ω^e. As shown in Figure 4.1, this linear operation can be represented by a matrix local to each element called the *multi-level extraction operator*.

Intuitively, such an operator flattens the multi-level hierarchy of functions into a sequence of elements equipped with a single-level basis. This benefit will become even more apparent when the multi-level extraction is combined with the Bézier extraction [Borden et al., 2011], as shortly discussed in Section 4.7 and illustrated in Figure 4.1. The multi-level Bézier extraction operator offers a classical finite element point of view on the hierarchical overlay of functions, giving each element the same set of functions in a reference space

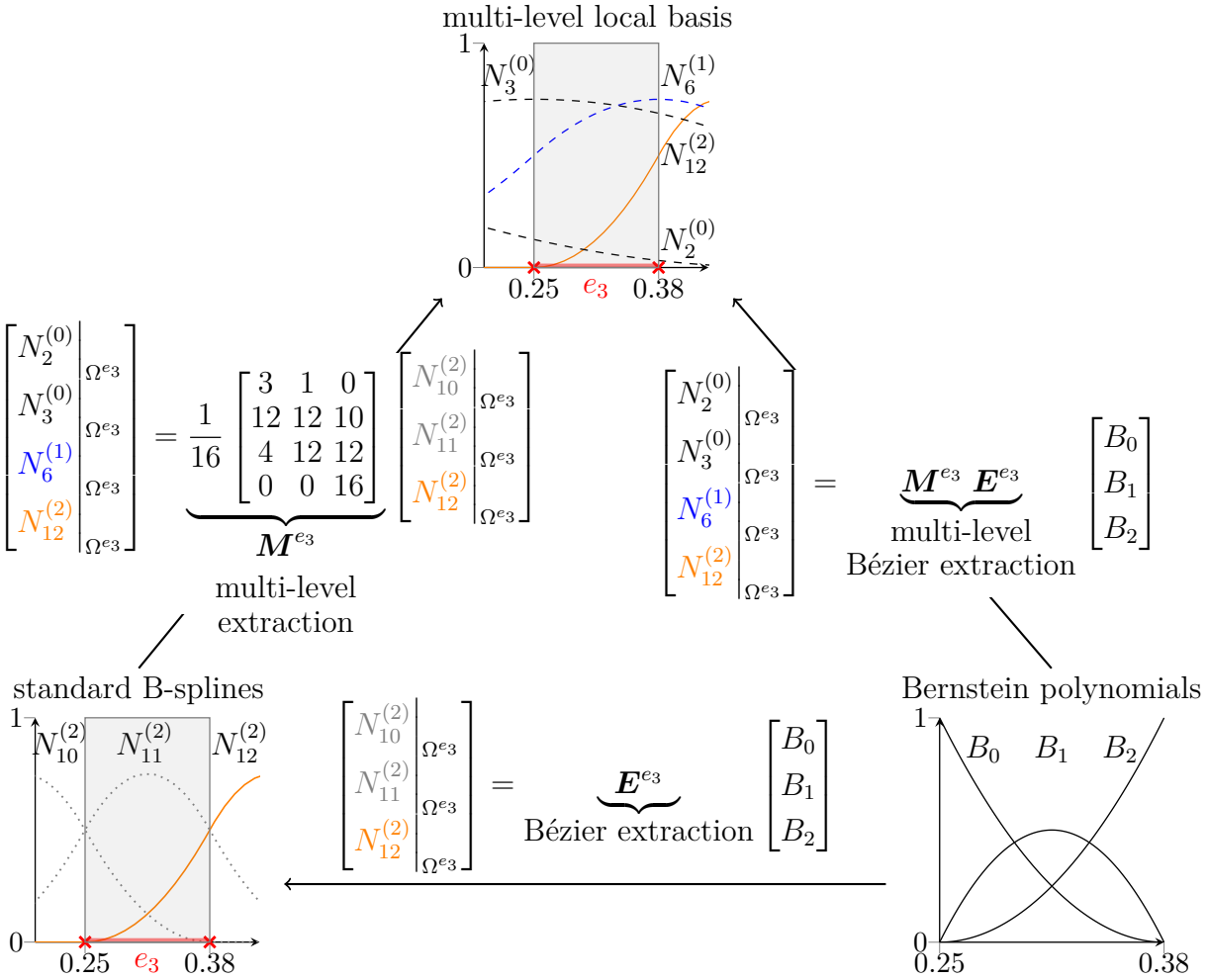


Figure 4.1: Hierarchical B-spline basis local to the element $\Omega^{e_3} = (0.25, 0.375)$ (cf. Figure 3.2), expressed as a linear combination of standard functions in $\mathcal{B}^{(2)}$ with support on Ω^{e_3} or of Bernstein polynomials.

that can generate the hierarchical basis. This representation can simplify the hierarchical-refinement implementation in already-existing finite element software.

This hierarchical-refinement representation can be defined not only for nested spline spaces produced by knot insertion, as the one considered in Figure 3.2, but it holds in a more general sense for every sequence of nested spaces $\mathcal{V}^{(0)} \subset \mathcal{V}^{(1)} \subset \cdots \subset \mathcal{V}^{(L)}$. In particular, different basis functions and different kind of refinements can be considered, e.g., (anisotropic) spline knot insertion, (anisotropic) spline degree elevation, (anisotropic) C^0-continuous linear polynomial and h-FEM refinement, or (anisotropic) C^0-continuous high-order polynomials and hp-FEM refinement (see, e.g., Di Stolfo et al. [2016]).

4.3 Representation of nested spaces

The core assumption in the formulation of the multi-level extraction operator is that the locally-refined basis functions are defined on nested function spaces. This assumption is

met in the case of $\mathcal{HB}$ and $\mathcal{THB}$, among others.

Let $\mathcal{V}^{(0)} \subset \mathcal{V}^{(1)} \subset \cdots \subset \mathcal{V}^{(L)}$ be a sequence of nested function spaces, with $\mathcal{B}^{(l)}$ being a (non necessarily B-spline) basis for $\mathcal{V}^{(l)}$, $m^{(l)} = dim(\mathcal{V}^{(l)})$, and $\boldsymbol{N}^{(l)}$ be the column vector composed of the $m^{(l)}$ functions in $\mathcal{B}^{(l)}$ in some fixed order. The spaces' nestedness implies that if $\boldsymbol{\tau} \in \mathcal{V}^{(l_1)}$, then $\boldsymbol{\tau} \in \mathcal{V}^{(l_2)}$ for any $l_2 \geq l_1$. Therefore, there exists a linear operator represented by a matrix $\boldsymbol{R}^{(l_1,l_2)} \in \mathbb{R}^{m^{(l_1)} \times m^{(l_2)}}$, such that

$$\boldsymbol{N}^{(l_1)} = \boldsymbol{R}^{(l_1,l_2)} \boldsymbol{N}^{(l_2)}, \qquad l_1 \leq l_2. \tag{4.1}$$

The operator $\boldsymbol{R}^{(l_1,l_2)}$ is referred to as *refinement operator*.

Furthermore, since $\boldsymbol{\tau} \in \mathcal{V}^{(l_1)}$, $\boldsymbol{\tau}$ can be written as $\boldsymbol{\tau} = (\boldsymbol{N}^{(l_1)})^\top \boldsymbol{P}^{(l_1)}$ for some coefficients $\boldsymbol{P}^{(l_1)}$. Moreover, since $\boldsymbol{\tau}$ belongs as well to $\mathcal{V}^{(l_2)}$, $\boldsymbol{\tau} = (\boldsymbol{N}^{(l_2)})^\top \boldsymbol{P}^{(l_2)}$ for some coefficients $\boldsymbol{P}^{(l_2)}$. Therefore, it holds

$$(\boldsymbol{N}^{(l_2)})^\top \boldsymbol{P}^{(l_2)} = \boldsymbol{\tau} = (\boldsymbol{N}^{(l_1)})^\top \boldsymbol{P}^{(l_1)} = (\boldsymbol{N}^{(l_2)})^\top (\boldsymbol{R}^{(l_1,l_2)})^\top \boldsymbol{P}^{(l_1)}. \tag{4.2}$$

Equation (4.2) implies

$$(\boldsymbol{N}^{(l_2)})^\top \left\{ \boldsymbol{P}^{(l_2)} - (\boldsymbol{R}^{(l_1,l_2)})^\top \boldsymbol{P}^{(l_1)} \right\} = 0,$$

and the linear independence of $\boldsymbol{N}^{(l_2)}$ yields the dual relation

$$\boldsymbol{P}^{(l_2)} = (\boldsymbol{R}^{(l_1,l_2)})^\top \boldsymbol{P}^{(l_1)}. \tag{4.3}$$

Equations (4.1) and (4.3) generalize Equations (2.35) and (2.34), respectively.

4.4 The multi-level extraction operator

Let us consider a hierarchical basis $\boldsymbol{H} = (H_1, .., H_m)^\top$ composed of functions taken from the level bases $\mathcal{B}^{(l)}$, $0 \leq l \leq L$, i.e.

$$\boldsymbol{H} = (N_1^{(0)}, .., N_{m^{(0)}}^{(0)}, .., N_1^{(L)}, .., N_{m^{(L)}}^{(L)})^\top.$$

Since each $H_k = N_i^{(l)} \in \mathcal{B}^{(l)}$ for some level l and index i, it follows from Equation (4.1) that each element H_k of $\boldsymbol{H}$ can be written as linear combination of basis functions $N_j^{(L)}$ in $\mathcal{B}^{(L)}$

$$H_k = N_i^{(l)} = \sum_{j=1}^{dim(\mathcal{B}^{(L)})} R_{ij}^{(l,L)} N_j^{(L)}.$$

Assembling all rows i of $\boldsymbol{R}^{(l,L)}$ corresponding to each hierarchical function H_k into a matrix $\boldsymbol{M}$, we obtain a representation of an operator mapping the finest-level functions $\boldsymbol{N}^{(L)}$ to the hierarchical basis $\boldsymbol{H}$

$$\boldsymbol{H} = \boldsymbol{M}\, \boldsymbol{N}^{(L)}.$$

The operator $\boldsymbol{M}$ is referred to as *multi-level extraction operator*. It is a generalization of the refinement operator $\boldsymbol{R}^{(l,L)}$, as it maps the functions of a single level to the hierarchical basis.

The multi-level extraction operator can be formally defined through the following two auxiliary definitions:

Definition 4.4.1: Restriction of a set of functions to a subdomain
Given Ω, let $\mathcal{A} = \{N_1, \dots, N_m\}$ be a set of scalar functions $N_i : \Omega \to \mathbb{R}$. Given a subdomain $\mathcal{Q} \subset \Omega$, the restriction $\operatorname{restr}(\mathcal{A}, \mathcal{Q})$ *of $\mathcal{A}$ to $\mathcal{Q}$ is*

$$\operatorname{restr}(\mathcal{A}, \mathcal{Q}) = \{N \in \mathcal{A} : \operatorname{supp}(N) \cap \mathcal{Q} \neq \emptyset\}.$$

Namely, it is the subset of functions with support on $\mathcal{Q}$.

Definition 4.4.2: Restrictions of a refinement operator
Let

$$\mathcal{B}^{(0)} = \{N_1^{(0)}, \dots, N_{m^{(0)}}^{(0)}\},$$
$$\mathcal{B}^{(1)} = \{N_1^{(1)}, \dots, N_{m^{(1)}}^{(1)}\}$$

be sets of functions $N_i^{(l)} : \Omega \to \mathbb{R}$. Let $\boldsymbol{N}^{(l)} = (N_1^{(l)}, \dots, N_{m^{(l)}}^{(l)})^\top$ be a vector composed of all functions in $\mathcal{B}^{(l)}$, $l \in \{0, 1\}$. Let $\boldsymbol{R}^{(0,1)}$ be a real $m^{(0)} \times m^{(1)}$ matrix such that

$$\boldsymbol{N}^{(0)} = \boldsymbol{R}^{(0,1)} \boldsymbol{N}^{(1)}.$$

Then, given $\mathcal{A}^{(0)} \subset \mathcal{B}^{(0)}$, $\mathcal{A}^{(1)} \subset \mathcal{B}^{(1)}$, the restriction $\operatorname{restr}(\boldsymbol{R}^{(0,1)}, \mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ *of $\boldsymbol{R}^{(0,1)}$ to $\mathcal{A}^{(0)}$ and $\mathcal{A}^{(1)}$ is the $|\mathcal{A}^{(0)}| \times |\mathcal{A}^{(1)}|$ submatrix obtained by selecting the rows with indices $r_1, \dots, r_{|\mathcal{A}^{(0)}|}$ and columns with indices $c_1, \dots, c_{|\mathcal{A}^{(1)}|}$ such that $N_{r_i}^{(0)} \in \mathcal{A}^{(0)}$ and $N_{c_j}^{(1)} \in \mathcal{A}^{(1)}$, $1 \leq i \leq |\mathcal{A}^{(0)}|$, $1 \leq j \leq |\mathcal{A}^{(1)}|$.*

Namely, $\operatorname{restr}(\boldsymbol{R}^{(0,1)}, \mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ *is the submatrix composed of rows associated with functions of $\mathcal{A}^{(0)}$ and columns associated with functions of $\mathcal{A}^{(1)}$.*

The row and column restrictions are defined as

$$\operatorname{rowr}(\boldsymbol{R}^{(0,1)}, \mathcal{A}^{(0)}) = \operatorname{restr}(\boldsymbol{R}^{(0,1)}, \mathcal{A}^{(0)}, \mathcal{B}^{(1)}),$$
$$\operatorname{colr}(\boldsymbol{R}^{(0,1)}, \mathcal{A}^{(1)}) = \operatorname{restr}(\boldsymbol{R}^{(0,1)}, \mathcal{B}^{(0)}, \mathcal{A}^{(1)}).$$

The multi-level extraction operator can be defined according to the following definition:

Definition 4.4.3: The multi-level extraction operator
Let

$$\mathcal{V}^{(0)} \subset \mathcal{V}^{(1)} \subset \cdots \subset \mathcal{V}^{(L)}$$

be a sequence of nested subspaces defined on a domain Ω. Let $\mathcal{B}^{(l)}$ be a basis of $\mathcal{V}^{(l)}$ and $\boldsymbol{N}^{(l)}$ a column vector of functions in $\mathcal{B}^{(l)}$. Furthermore, let $\boldsymbol{R}^{(l_1,l_2)}$ be the refinement operator such that

$$\boldsymbol{N}^{(l_1)} = \boldsymbol{R}^{(l_1,l_2)} \boldsymbol{N}^{(l_2)}, \qquad l_1 \leq l_2.$$

Given $\mathcal{H} \subset \bigcup_l \mathcal{B}^{(l)}$ (e.g., $\mathcal{H} = \mathcal{HB}$ or $\mathcal{H} = \mathcal{THB}$), let

$$\mathcal{H}^{(l)} = \mathcal{H} \cap \mathcal{B}^{(l)}$$

be the level-l functions in $\mathcal{H}$.

The global multi-level operator $\boldsymbol{M}$ is defined as the matrix

$$\boldsymbol{M} = \begin{bmatrix} \text{rowr}\left(\boldsymbol{R}^{(0,L)}, \mathcal{H}^{(0)}\right) \\ \vdots \\ \text{rowr}\left(\boldsymbol{R}^{(0,L)}, \mathcal{H}^{(L)}\right) \end{bmatrix}.$$

Furthermore, given an element subdomain $\Omega^e \subset \Omega$ of level l^e, the local *multi-level extraction operator $\boldsymbol{M}^e$ is defined as*

$$\boldsymbol{M}^e = \begin{bmatrix} \boldsymbol{R}^{e,(0,l^e)} \\ \vdots \\ \boldsymbol{R}^{e,(l^e,l^e)} \end{bmatrix}, \qquad \boldsymbol{R}^{e,(l,l^e)} = \text{restr}\left(\boldsymbol{R}^{(l,l^e)}, \mathcal{H}^{e,(l)}, \mathcal{B}^{e,(l^e)}\right)$$

where $\mathcal{H}^{e,(l)} = \text{restr}(\mathcal{H}^{e,(l)}, \Omega^e)$, and $\mathcal{B}^{e,(l)} = \text{restr}(\mathcal{B}^{(l)}, \Omega^e)$.

Namely, $\boldsymbol{M}^e$ is the submatrix obtained from $\boldsymbol{M}$ by selecting the rows and columns associated with functions having support on Ω^e.

The global multi-level extraction operator $\boldsymbol{M}$ is simply obtained by joining the rows of the operators $\boldsymbol{R}^{(l,L)}$, $l = 0, \dots, L$, associated with the hierarchical basis $\mathcal{H}$. It allows representing the target basis $\mathcal{H}$ in terms of the standard functions of the finest level. Indeed, denoting by $\boldsymbol{H}^{(l)}$ the column vector of functions in $\mathcal{H}^{(l)}$ sorted consistently with $\mathit{rowr}(\boldsymbol{R}^{(0,L)}, \mathcal{H}^{(0)})$, it holds

$$\begin{aligned} \boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}^{(0)} \\ \vdots \\ \boldsymbol{H}^{(L)} \end{bmatrix} &= \begin{bmatrix} \mathit{rowr}(\boldsymbol{R}^{(0,L)}, \mathcal{H}^{(0)})\ \boldsymbol{N}^{(L)} \\ \vdots \\ \mathit{rowr}(\boldsymbol{R}^{(L,L)}, \mathcal{H}^{(L)})\ \boldsymbol{N}^{(L)} \end{bmatrix} \\ &= \begin{bmatrix} \mathit{rowr}(\boldsymbol{R}^{(0,L)}, \mathcal{H}^{(0)}) \\ \vdots \\ \mathit{rowr}(\boldsymbol{R}^{(L,L)}, \mathcal{H}^{(L)}) \end{bmatrix} \boldsymbol{N}^{(L)} \\ &= \boldsymbol{M}\ \boldsymbol{N}^{(L)}. \end{aligned}$$

Furthermore, for $0 \le l \le L$, one can map $\boldsymbol{N}^{(l)}$ to the hierarchical functions up to level l through a multi-level extraction operator $\boldsymbol{M}^{(l)}$.

$$\begin{bmatrix} \boldsymbol{H}^{(0)} \\ \vdots \\ \boldsymbol{H}^{(l)} \end{bmatrix} = \begin{bmatrix} \mathit{rowr}(\boldsymbol{R}^{(0,l)}, \mathcal{H}^{(0)}) \\ \vdots \\ \mathit{rowr}(\boldsymbol{R}^{(l,l)}, \mathcal{H}^{(l)}) \end{bmatrix} \boldsymbol{N}^{(l)} = \boldsymbol{M}^{(l)}\ \boldsymbol{N}^{(l)}.$$

The multi-level operator $\boldsymbol{M}$ provides the dual relation analogous to Equation (4.3). Indeed, using the same reasoning as in Chapter 4 and section 4.3, it holds

$$(\boldsymbol{N}^{(L)})^\top \left\{ \boldsymbol{P}^{(L)} - \boldsymbol{M}^\top \boldsymbol{P}^{(\boldsymbol{H})} \right\} = 0,$$

for coefficients $\boldsymbol{P}^{(\boldsymbol{H})}$ and $\boldsymbol{P}^{(L)}$ such that $\boldsymbol{H}^\top \boldsymbol{P}^{(\boldsymbol{H})} = \boldsymbol{N}^\top \boldsymbol{P}^{(L)}$. The linear independence of $(\boldsymbol{N}^{(L)})^\top$ yields

$$\boldsymbol{P}^{(L)} = \boldsymbol{M}^\top \ \boldsymbol{P}^{(\boldsymbol{H})}, \tag{4.4}$$

The local multi-level extraction operator $\boldsymbol{M}^e$ associated with a level-l^e element Ω^e is obtained by extracting the smallest sub-matrix affecting Ω^e, i.e., selecting rows and columns corresponding to functions with support on Ω^e. It allows representing the local basis $\mathcal{H}^e$ in terms of the standard level-l^e functions. In particular, letting $\boldsymbol{H}^{e,(l^e)}$ and $\boldsymbol{N}^{e,(l^e)}$ be appropriate column vectors of functions in $\mathcal{H}^{e,(l^e)}$ and $\mathcal{B}^{e,(l^e)}$, respectively, it holds

$$\begin{aligned}
\boldsymbol{H}^e(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{H}^{e,(0)}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{H}^{e,(l^e)}(\boldsymbol{x}) \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R}^{e,(0,l^e)} \ \boldsymbol{N}^{e,(l^e)}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{R}^{e,(l^e,l^e)} \ \boldsymbol{N}^{e,(l^e)}(\boldsymbol{x}) \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{R}^{e,(0,l^e)} \\ \vdots \\ \boldsymbol{R}^{e,(l^e,l^e)} \end{bmatrix} \boldsymbol{N}^{e,(l^e)}(\boldsymbol{x}) \\
&= \boldsymbol{M}^e \ \boldsymbol{N}^{e,(l^e)}(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \Omega^e.
\end{aligned} \tag{4.5}$$

Note that $\boldsymbol{M}^e$ maps the functions $\boldsymbol{N}^{e,(l^e)}$ belonging to the same level l^e as the element.

The dual relation (4.4) also holds locally. In particular,

$$\boldsymbol{P}^{e,(l^e)} = (\boldsymbol{M}^e)^\top \boldsymbol{P}^{e,(\boldsymbol{H}^e)} \tag{4.6}$$

for some coefficients $\boldsymbol{P}^{e,(\boldsymbol{H}^e)}$, and $\boldsymbol{P}^{e,(l^e)}$, such that $(\boldsymbol{H}^e)^\top \boldsymbol{P}^{e,(\boldsymbol{H}^e)} = (\boldsymbol{N}^{e,(l^e)})^\top \boldsymbol{P}^{e,(l^e)}$ on Ω^e.

Equation (4.6) can be used for translating degrees of freedom (for the solution description) or control points (for the geometry description) between the local hierarchical basis $\boldsymbol{H}^e$ and the single-level basis $\boldsymbol{N}^{e,(l^e)}$. A similar interpretation can be drawn for the corresponding global operation in Equation (4.4). The above relations generalize the local and global version of the properties of the Bézier extraction given in Equations (2.35) and (2.34) and in Borden et al. [2011].

In the following sections, the multi-level extraction operator's general definition is applied to the (truncated) hierarchical B-splines.

4.5 Extraction of hierarchical B-splines

The multi-level extraction can be applied to the hierarchical basis $\mathcal{H} = \mathcal{HB}$ associated with a sequence of nested spline spaces

$$\mathcal{V}^{(0)} \subset \mathcal{V}^{(1)} \subset \cdots \subset \mathcal{V}^{(L)}$$

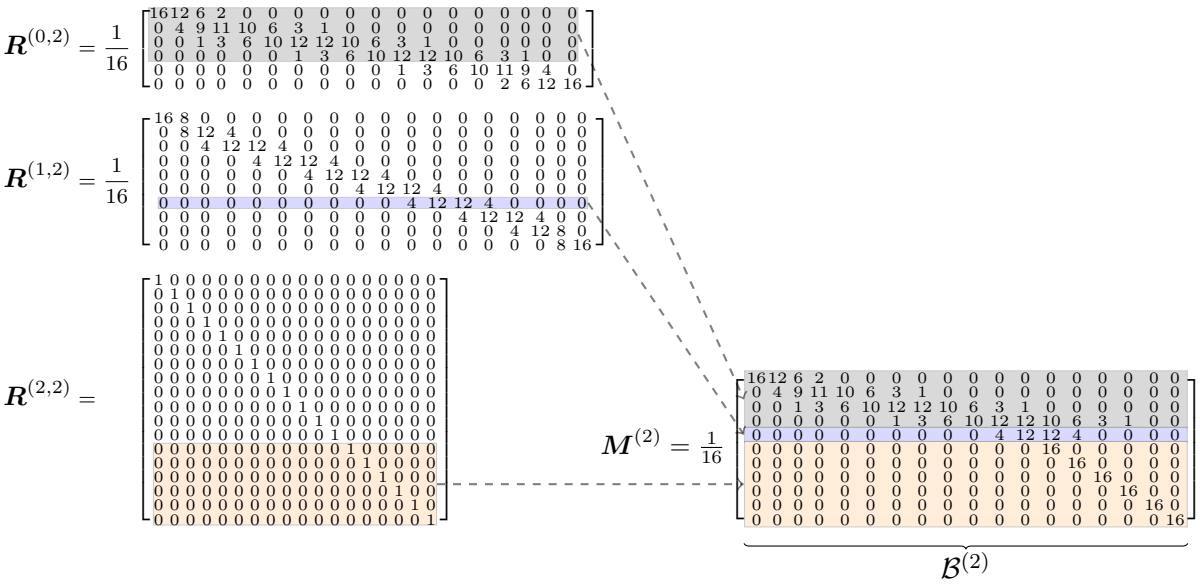
of fixed degree p obtained by knot insertion. Let $\mathcal{B}^{(l)}$ be the B-spline basis of $\mathcal{V}^{(l)}$. Then the refinement operator $\boldsymbol{R}^{(l_1,l_2)}$ in Equation (4.1) is known from the literature (see, e.g., Boehm [1985]), and it is referred to as the *knot-insertion operator*.

Its entries can be calculated by standard knot-insertion techniques such as Boehm's or Oslo's algorithm [Boehm, 1985]. For example, the following picture shows the operator $\boldsymbol{R}^{(0,2)}$ for Figure 3.2

$$\boldsymbol{R}^{(0,2)} = \frac{1}{16}\begin{bmatrix}
16 & 12 & 6 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 4 & 9 & 11 & 10 & 6 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 & 6 & 10 & 12 & 12 & 10 & 6 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 6 & 10 & 12 & 12 & 10 & 6 & 3 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 6 & 10 & 11 & 9 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6 & 12 & 16
\end{bmatrix}
\begin{matrix}
\leftarrow N_0^{(0)} \\ \leftarrow N_1^{(0)} \\ \leftarrow N_2^{(0)} \\ \leftarrow N_3^{(0)} \\ \leftarrow N_4^{(0)} \\ \leftarrow N_5^{(0)}
\end{matrix}
\begin{matrix}
\} \mathcal{B}^{(0)} \\ \} \mathcal{H}^{(0)}
\end{matrix}$$

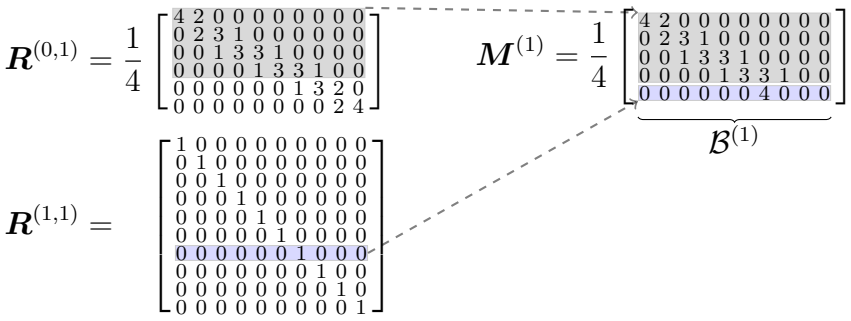
$$\underbrace{\begin{matrix}
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
N_0^{(2)} & N_1^{(2)} & N_2^{(2)} & N_3^{(2)} & N_4^{(2)} & N_5^{(2)} & N_6^{(2)} & N_7^{(2)} & N_8^{(2)} & N_9^{(2)} & N_{10}^{(2)} & N_{11}^{(2)} & N_{12}^{(2)} & N_{13}^{(2)} & N_{14}^{(2)} & N_{15}^{(2)} & N_{16}^{(2)} & N_{17}^{(2)}
\end{matrix}}_{\mathcal{B}^{(2)}}$$

According to Definition 4.4.3, the global multi-level extraction operator $\boldsymbol{M}^{(L)}$ is obtained by joining the rows of the operators $\boldsymbol{R}^{(l,L)}$, $l = 0, \ldots, L$, associated with the hierarchical B-splines basis functions, as illustrated below.



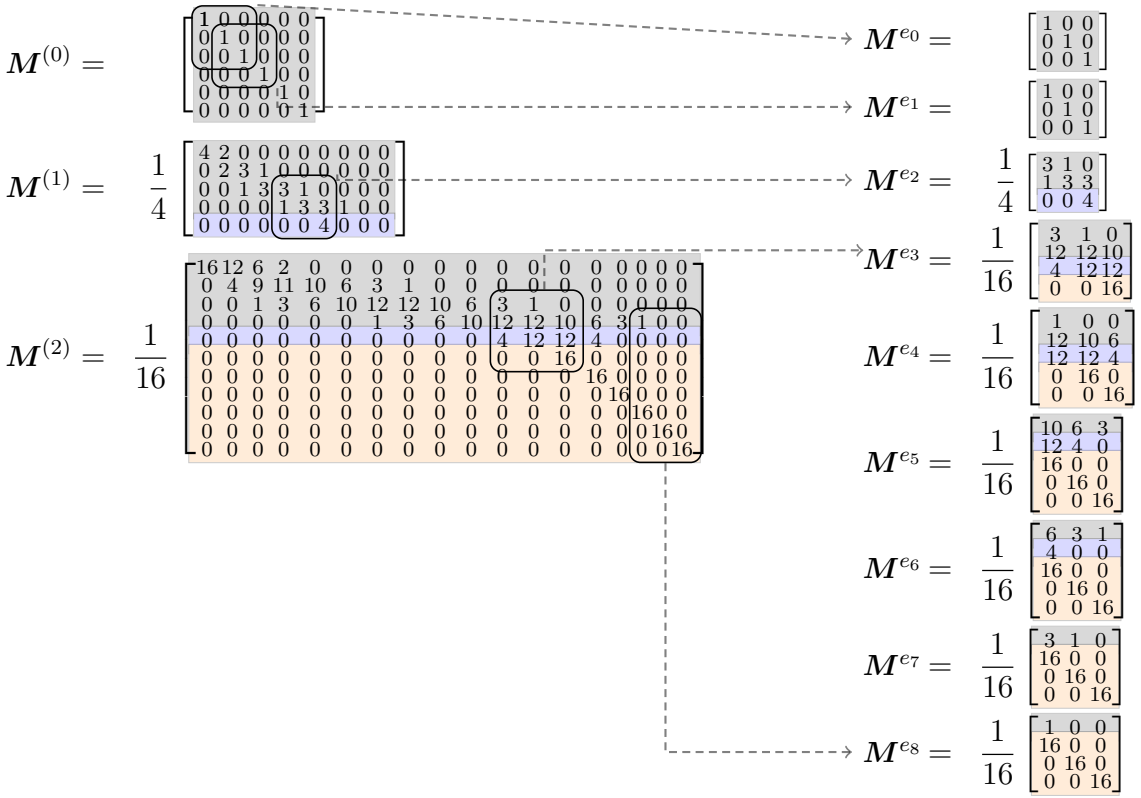
The highlighted rows of $\boldsymbol{R}^{(0,2)}$ correspond to the functions $\mathcal{HB}^{(0)} = \{N_1^{(0)}, ..., N_4^{(0)}\}$, the highlighted row of $\boldsymbol{R}^{(1,2)}$ is associated with $\mathcal{HB}^{(1)} = \{N_6^{(1)}\}$, and the highlighted rows of $\boldsymbol{R}^{(2,2)}$ correspond to $\mathcal{HB}^{(2)} = \{N_{12}^{(2)}, N_{13}^{(2)}, \ldots, N_{17}^{(2)}\}$.

A similar picture can be drawn for $\boldsymbol{M}^{(1)}$, as shown below.



4.5.1 Element extraction of hierarchical B-splines

The global multi-level extraction operator $\boldsymbol{M}^{(l^e)}$ can be localized to each level-l^e element Ω^e by selecting the rows and columns associated with functions with support on Ω^e. Note that the multi-level extraction operator is always of the same level l^e as the element. Considering again the example in Figure 3.2, the element can be constructed as in the following picture.



Note that the operator $\boldsymbol{M}^{e_3}$ is the same as the one previously shown in Figure 4.1. The localization of the multi-level extraction operator is analogous to the Bézier extraction operator's localization discussed in Borden et al. [2011].

4.6 Extraction of truncated hierarchical B-splines

Following [Giannelli et al., 2016, 2012], it can be deduced directly from Definitions 3.4.1 and 3.4.2 that for the truncated basis, the refinement operators $trunc(\boldsymbol{R}^{(l,l+1)})$ of consecutive levels can be obtained from the standard knot insertion operators $\boldsymbol{R}^{(l,l+1)}$ (see Section 4.5) as follows

$$\left[trunc\left(\boldsymbol{R}^{(l,l+1)}\right)\right]_{ij} = \begin{cases} R_{ij}^{(l,l+1)} & \text{if } N_j^{(l+1)} \in \mathcal{B}^{(l+1)} \wedge supp(N_j^{(l+1)}) \cap \Omega_-^{l+1} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

The truncated refinement operators across multiple levels $Trunc(\boldsymbol{R}^{(l_1,l_2)})$, $l_2 \geq l_1$, can be defined recursively as follows

$$Trunc(\boldsymbol{R}^{(l_1,l_2)}) = \begin{cases} \boldsymbol{I} & \text{if } l_2 = l_1, \\ trunc\left(\boldsymbol{R}^{(l_1,l_1+1)}\right) & \text{if } l_2 = l_1 + 1, \\ trunc\left(\boldsymbol{R}^{(l_1,l_1+1)}\right) \; Trunc\left(\boldsymbol{R}^{(l_1+1,l_2)}\right) & \text{otherwise,} \end{cases}$$

where $\boldsymbol{I}$ is the identity matrix of size $|\mathcal{B}^{(l_2)}| \times |\mathcal{B}^{(l_2)}|$.

The global and local multi-level extraction operators for the truncated hierarchical B-splines are obtained by Definition 4.4.3 using the refinement operators $trunc(\boldsymbol{R}^{(l_1,l_2)})$.

4.6.1 Element extraction of truncated hierarchical B-splines

Note that the local truncated operator $Trunc(\boldsymbol{R}^{e,(l_1,l_2)})$ of a level-l^e element Ω^e can also be defined constructively directly by recursive truncation of local operators $\boldsymbol{R}^{e,(l,l+1)}$ between consecutive levels. Namely,

$$\left[trunc\left(\boldsymbol{R}^{e,(l,l+1)}\right)\right]_{ij} = \begin{cases} R_{ij}^{e,(l,l+1)} & \text{if } N_j^{(l+1)} \in \mathcal{B}^{e,(l+1)} \wedge N_j^{(l+1)} \cap \Omega_-^{l+1} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

$$Trunc(\boldsymbol{R}^{e,(l_1,l_2)}) = \begin{cases} \boldsymbol{I} & \text{if } l_2 = l_1, \\ trunc\left(\boldsymbol{R}^{e,(l_1,l_1+1)}\right) & \text{if } l_2 = l_1 + 1, \\ trunc\left(\boldsymbol{R}^{e,(l_1,l_1+1)}\right) \; Trunc\left(\boldsymbol{R}^{e,(l_1+1,l_2)}\right) & \text{otherwise,} \end{cases} \tag{4.7}$$

For example, considering again Ω^{e_3} in Figure 3.2, $\boldsymbol{R}^{e_3,(1,2)}$ and $\boldsymbol{R}^{e_3,(0,1)}$ can be seen directly as submatrices of the operators shown in Section 4.5:

$$\boldsymbol{R}^{e_3,(0,1)} = \frac{1}{4}\begin{bmatrix} 1 & 0 & 0 \\ 3 & 3 & 1 \\ 0 & 1 & 3 \end{bmatrix} \qquad \boldsymbol{R}^{e_3,(1,2)} = \frac{1}{4}\begin{bmatrix} 3 & 1 & 0 \\ 1 & 3 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$trunc\left(\boldsymbol{R}^{e_3,(0,1)}\right) = \frac{1}{4}\begin{bmatrix} 1 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad trunc\left(\boldsymbol{R}^{e_3,(1,2)}\right) = \frac{1}{4}\begin{bmatrix} 3 & 1 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

After computing the recursively-truncated local refinement operators

$$
\begin{aligned}
Trunc(\boldsymbol{R}^{e_3,(2,2)}) = \boldsymbol{I} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
Trunc(\boldsymbol{R}^{e_3,(1,2)}) = trunc\left(\boldsymbol{R}^{e_3,(1,2)}\right) &= \frac{1}{16}\begin{bmatrix} 12 & 4 & 0 \\ 4 & 12 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
Trunc(\boldsymbol{R}^{e_3,(0,2)}) = trunc\left(\boldsymbol{R}^{e_3,(0,1)}\right)\ trunc\left(\boldsymbol{R}^{e_3,(1,2)}\right) &= \frac{1}{16}\begin{bmatrix} 3 & 1 & 0 \\ 9 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix},
\end{aligned}
$$

the multi-level extraction operator can be assembled the usual way

$$
\boldsymbol{M}^{e_3} = \frac{1}{16}\begin{bmatrix} 3 & 1 & 0 \\ 9 & 3 & 0 \\ 4 & 12 & 0 \\ 0 & 0 & 16 \end{bmatrix}
$$

Note that the recursion in Equation (4.7) shows an alternative direct way to compute $\boldsymbol{M}^e = \boldsymbol{M}^{e,(l)}$

$$
\begin{aligned}
\boldsymbol{M}^{e,(1)} &= \boldsymbol{J}^{e,(1)}, \\
\boldsymbol{M}^{e,(l+1)} &= \begin{bmatrix} \boldsymbol{M}^{e,(l)} & Trunc\left(\boldsymbol{R}^{e,(l,l+1)}\right) \\ & \boldsymbol{J}^{e,(l+1)} \end{bmatrix}, \qquad l = 1, \ldots, l^e - 1
\end{aligned}
$$

where $\boldsymbol{J}^{e,(l)}$ is a matrix that selects the element active functions of level l

$$
J_{ij}^{e,(l)} = \begin{cases} 1 & \text{if } H_i^{e,(l)} = N_j^{e,(l)}, \\ 0 & \text{otherwise.} \end{cases}
$$

For example, for element Ω^{e_3}:

$$
\boldsymbol{J}^{e_3,(0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{J}^{e_3,(1)} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{J}^{e_3,(2)} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}
$$

$$
\boldsymbol{M}^{e_3,(0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{M}^{e_3,(1)} = \frac{1}{4}\begin{bmatrix} 1 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 4 & 0 \end{bmatrix} \qquad \boldsymbol{M}^{e_3,(2)} = \frac{1}{16}\begin{bmatrix} 3 & 1 & 0 \\ 9 & 3 & 0 \\ 4 & 12 & 0 \\ 0 & 0 & 16 \end{bmatrix}
$$

The algorithm is the local version of the one presented in [Garau and Vázquez, 2016, 2018], showing that the same algorithm can build local operators. If the truncation operator $Trunc(\cdot)$ is removed, the above algorithm can be used to define the local multi-level extraction operators for the hierarchical B-splines through local computations.

4.7 The multi-level Bézier extraction

The multi-level extraction operator $\boldsymbol{M}^e$ can be combined with the Bézier extraction operator $\boldsymbol{E}^e$ defined in Section 2.6.1, as depicted in Figure 4.1. This combination creates a direct map $\boldsymbol{C}^e$ from a standard set of reference basis functions $\boldsymbol{B}$ to the hierarchical basis $\boldsymbol{H}^e$

$$
\begin{aligned}
\boldsymbol{H}^e &= \boldsymbol{M}^e \boldsymbol{N}^{e,(l^e)} \\
&= \boldsymbol{M}^e \boldsymbol{E}^e \boldsymbol{B} \\
&= \boldsymbol{C}^e \boldsymbol{B}.
\end{aligned} \tag{4.8}
$$

Similarly, given some coefficients $\boldsymbol{P}^e$, the coefficients $\boldsymbol{Q}^e$ such that $\boldsymbol{B}^\top \boldsymbol{Q}^e = (\boldsymbol{H}^e)^\top \boldsymbol{P}^e$ can be obtained as

$$\boldsymbol{Q}^e = (\boldsymbol{C}^e)^\top \boldsymbol{P}^e. \tag{4.9}$$

The reference basis functions $\boldsymbol{B}$ are the same for every element, allowing the hierarchical refinement to be handled in a very similar way to standard finite element implementations. A direct geometric mapping in terms of the Bernstein functions can be obtained through the control points $\boldsymbol{Q}^e$ in Equation (4.9).

Note that the Bézier extraction is based on Bernstein polynomials, as they naturally arise from the knot repetition procedure. However, any other polynomial basis can be employed, such as the Lagrange polynomials, combining the multi-level operator with the Lagrange extraction presented in Schillinger et al. [2016].

It should be noted that the hierarchical refinement introduces a non-constant number of degrees of freedom per element. For example, consider element e_0 (3 DOFs) and element e_4 (5 DOFs) in Figure 3.2. Although the multi-level Bézier extraction can significantly ease the introduction of hierarchical refinement in standard finite element implementations, the existing code should still allow for element matrices of non-constant size when assembling the system matrices. This requirement seems to be inherent to the method.

4.8 Extension to higher dimensional spaces

The previous sections' definitions are directly valid also for multi-dimensional spaces. In particular, the assumption in Section 4.3 is also met for higher-dimensional spaces. Namely, there exists a refinement operator $\boldsymbol{R}^{(l_1,l_2)}$ between the spaces $\mathcal{V}^{(l_1)}$ and $\mathcal{V}^{(l_2)}$ that can be used to define the multi-level extraction, as in Section 4.4.

For tensor-product spaces, the refinement operator $\boldsymbol{R}^{(l_1,l_2)}$ can be computed as the Kronecker product of the univariate refinement operators. Moreover, according to Section 4.4, just some rows of $\boldsymbol{R}^{(l_1,l_2)}$ are needed. Therefore, the Kronecker product can be limited to the necessary rows. The same structure also applies for the element-localizations $\boldsymbol{R}^{e,(l_1,l_2)}$. However, Chapter 5 proposes a strategy where the tensor-product operators do not need to be explicitly constructed.

It is worth noting that the hierarchical basis $\boldsymbol{H}$ is, in general, not of tensor-product structure. Therefore, there is no possibility of applying the multi-level extraction operator to each parametric direction and constructing $\boldsymbol{H}$ by taking the tensor product of (extracted) univariate functions. Leveraging the tensor product is not possible even for the local basis defined on an element's reference space. Such optimization is possible with the standard Bézier extraction for non-hierarchical meshes [Borden et al., 2011], but it does not apply directly to hierarchical refinement. However, a strategy to leverage the tensor structure is proposed in Chapter 5.

4.9 Extraction of hierarchical NURBS

The extraction of (truncated) hierarchical B-splines can be extended to a hierarchical definition of NURBS functions through homogeneous coordinates. In particular, let

$$\mathring{\boldsymbol{F}} = \sum_i \mathring{\boldsymbol{P}}_i^{(0)} \, N_i^{(0)} \tag{4.10}$$

be a NURBS geometry defined by the base-level B-splines, $\boldsymbol{N}^{(0)}$, and by the control points $\mathring{\boldsymbol{P}}^{(0)}$ in homogeneous coordinates with respect to the weights $\boldsymbol{w}^{(0)}$. By standard knot-insertion, $\mathring{\boldsymbol{F}}$ can be written in terms of a hierarchical basis $\boldsymbol{H}$

$$\sum_i \mathring{\boldsymbol{P}}_i^{(\boldsymbol{H})} \, H_i = \mathring{\boldsymbol{F}} = \sum_i \mathring{\boldsymbol{P}}_i^{(0)} \, N_i^{(0)} \tag{4.11}$$

according to the usual relations

$$\boldsymbol{N}^{(0)} = \boldsymbol{L}\boldsymbol{H}, \tag{4.12}$$

$$\mathring{\boldsymbol{P}}^{(\boldsymbol{H})} = \boldsymbol{L}^\top \mathring{\boldsymbol{P}}^{(0)}. \tag{4.13}$$

Equation (4.13) indicates that the hierarchical weights $\boldsymbol{w}^{(\boldsymbol{H})}$ associated with the hierarchical basis $\boldsymbol{H}$ are obtained as

$$\boldsymbol{w}^{(\boldsymbol{H})} = \boldsymbol{L}^\top \boldsymbol{w}^{(0)}. \tag{4.14}$$

Similarly, $\mathring{\boldsymbol{F}}$ can be written in terms of the finest level L

$$\mathring{\boldsymbol{F}} = \sum_i \mathring{\boldsymbol{P}}_i^{(L)} \, N_i^{(L)}, \tag{4.15}$$

with

$$\boldsymbol{H} = \boldsymbol{M}\boldsymbol{N}^{(L)}, \tag{4.16}$$

$$\mathring{\boldsymbol{P}}^{(L)} = \boldsymbol{M}^\top \mathring{\boldsymbol{P}}^{(\boldsymbol{H})}. \tag{4.17}$$

Therefore, the hierarchical B-splines $\boldsymbol{H}$ can be obtained by multi-level extraction of the finest level B-splines, as defined in Equations (4.16) and (4.17) and described in Section 4.5. The hierarchical NURBS functions representing the original geometry are obtained through the weights $\boldsymbol{w}^{(\boldsymbol{H})}$ as

$$\bar{R}_i = \frac{w_i^{(\boldsymbol{H})} \, H_i}{\sum_j w_j^{(\boldsymbol{H})} \, H_j}. \tag{4.18}$$

4.10 Extension to other sequences of nested spaces

In general, the multi-level extraction operator can be constructed for every sequence of nested spaces. Indeed the nestedness assumption ensures the existence of the refinement operator $\boldsymbol{R}^{(l_1,l_2)}$: the fundamental component of the definitions in Section 4.4.

The construction of the refinement operator $\boldsymbol{R}^{(l_1,l_2)}$ depends on the specific refinement procedure. In the previous sections, the whole approach was exemplified for the classical (truncated) hierarchical B-splines produced by knot insertion. In this case, $\boldsymbol{R}$ can be computed by standard knot insertion techniques, and Chapter 5 shows a few algorithms to construct the univariate element-localized refinement operators.

Other examples of refinement strategies defined on nested spaces are (anisotropic) C^0-continuous linear polynomial and h-FEM refinement, or (anisotropic) C^0-continuous high-order polynomials and hp-FEM refinement (see, e.g., Di Stolfo et al. [2016]; Zander et al. [2016, 2015]). The following section shows a further example in the context of degree elevation.

4.10.1 Outlook: the multi-level extraction for degree elevation

The refinement operator $\boldsymbol{R}^{(l_1,l_2)}$ for degree elevation can be constructed using standard degree-elevation techniques (see, e.g., Lee and Park [2000]; Piegl and Tiller [1995]).

For example, consider the B-splines $\boldsymbol{N}^{(0)}$ of degree $p^{(0)} = 2$ and $\boldsymbol{N}^{(1)}$ of degree $p^{(1)} = 3$, respectively defined by the open knot vectors

$$\boldsymbol{\Xi}^{(0)} = (-1, -1, -1 - 0.5, 0, 0.75, 1, 1, 1),$$
$$\boldsymbol{\Xi}^{(1)} = (-1, -1, -1, -1, -0.5, -0.5, 0, 0, 0.75, 0.75, 1, 1, 1, 1).$$

Figure 4.2 shows the corresponding B-splines' overlay, together with a possible choice for the hierarchical functions $\boldsymbol{H}$. For this example, the refinement operator $\boldsymbol{R}^{(0,1)}$ and multi-level extraction operator $\boldsymbol{M}^{(1)}$ are defined as in the following picture.

$$\boldsymbol{R}^{(0,1)} = \frac{1}{60}\begin{bmatrix} 60 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 50 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 50 & 52 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 48 & 45 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 & 55 & 40 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 60 \end{bmatrix} \qquad \boldsymbol{M} = \frac{1}{60}\begin{bmatrix} 60 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 50 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 50 & 52 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 48 & 45 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 \end{bmatrix}$$

Instead, the local operators read as follows.

$$\boldsymbol{M} = \frac{1}{60}\begin{bmatrix} 60 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 50 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 50 & 52 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 48 & 45 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 60 \end{bmatrix}$$

$$\boldsymbol{M}^{e_0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M}^{e_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M}^{e_2} = \frac{1}{60}\begin{bmatrix} 52 & 12 & 0 & 0 \\ 8 & 48 & 45 & 5 \\ 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 60 \end{bmatrix}$$

$$\boldsymbol{M}^{e_3} = \frac{1}{60}\begin{bmatrix} 45 & 5 & 0 & 0 \\ 60 & 0 & 0 & 0 \\ 0 & 60 & 0 & 0 \\ 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 60 \end{bmatrix}$$

Note that $\boldsymbol{M}^{e_0}$ and $\boldsymbol{M}^{e_1}$ are composed of three columns, while $\boldsymbol{M}^{e_2}$ and $\boldsymbol{M}^{e_3}$ have four columns. This difference corresponds to the fact that the e_0 and e_1 belong to the 0th level, where $\mathcal{B}^{(0)}$ has degree $p^{(0)} = 2$, while e_2 and e_3 belong to level 1, having cubic basis functions.

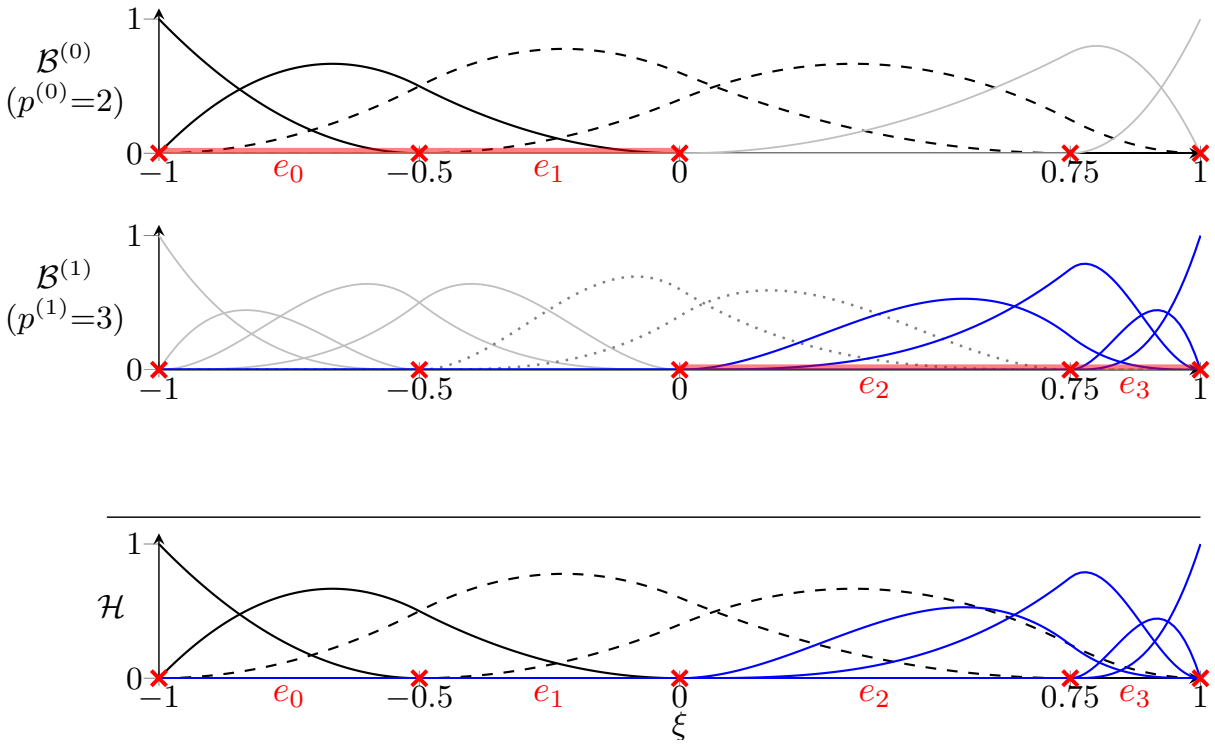


Figure 4.2: Example of hierarchical B-splines of a different order. The knots of each level are marked by red crosses.

Chapter 5

Algorithms for the multi-level Bézier extraction

Chapter 4 formulated the multi-level extraction operator for local refinement based on any sequence of nested spaces, focusing on its application to (truncated) hierarchical B-splines. The primary motivation for developing these approaches is to perform local refinement that cannot be obtained by tensor products. On the other hand, such a tensor-product structure could be leveraged to formulate efficient extraction algorithms. This chapter attempts to conciliate these two aspects by proposing an approach where:

- the univariate extraction-operators are computed once for each parametric direction and hierarchical level,
- the following operations are formulated using only the univariate operators, without explicitly constructing and storing the full tensor-product operator:
  - extraction of functions,
  - degree-of-freedom projection for refinement, and
  - degree-of-freedom projection for coarsening (modified Bézier projection).

5.1 Introduction

In the literature, tensor factorization is combined with extraction and projection of degrees of freedom (DOFs) for meshes that are not hierarchically refined. In the original Bézier extraction publication [Borden et al., 2011], basis functions are extracted in each parametric direction of a (non-hierarchically refined) patch. In Thomas et al. [2015], the projection of DOFs between two (non-hierarchically refined) tensor-product patches is performed through a sequence of univariate multiplications where possible, but a Kronecker product is ultimately computed. In this chapter, the tensor-product structure is further exploited through algorithms to multiply by a Kronecker matrix without explicitly computing the Kronecker product. Extraction and projection are extended to hierarchical refinement in several works [Apprich et al., 2014; Bornemann and Cirak, 2013; Evans et al., 2015; Garau and Vázquez, 2018; Hennig et al., 2018, 2016; Lorenzo et al., 2017; Scott et al., 2014; Vuong, 2014], but tensor-product factorization is not discussed. In

[Bressan and Mokriš, 2017], a general function extraction framework is presented for various local refinement strategies. The function-extraction algorithms proposed in this work can be seen as an iterative version of their approach. However, the projection of DOFs is not discussed in Bressan and Mokriš [2017]. The projection of DOFs for mesh refinement proposed in this work can be seen as an iterative version of the algorithms developed in Garau and Vázquez [2018]. Finally, the spline evaluation presented in Giannelli et al. [2016] can be seen as a DOF projection to the most refined level and a particular case of the DOF projection algorithm between two hierarchical meshes.

This work focuses on the element point of view to facilitate the implementation of local refinement in already-existing software. If the code design allows it, leveraging the whole patch tensor-product structure (instead of looping over the knot spans) further increases efficiency (see, e.g., Calabrò et al. [2019]; Calabrò et al. [2017]; Hiemstra et al. [2019]; Sangalli and Tani [2018]). However, combining this approach to local refinement and trimming is still an active area of research.

5.2 Construction of univariate knot insertion refinement operator

This section proposes a few general procedures to construct the knot-insertion operators. These algorithms are inspired by Lyche and Morken [2008]; Piegl and Tiller [1995] and are expressed in the MATLAB-like syntax [MATLAB, 2019].

Each pair $(\mathcal{V}^{(1)}, \mathcal{V}^{(2)})$ of univariate nested B-spline spaces, $\mathcal{V}^{(1)} \subset \mathcal{V}^{(2)}$, is defined by some knot vectors

$$\Xi^{(l)} = (\xi_1^{(l)}, \dots, \xi_{m^{(l)}+p+1}^{(l)}), \qquad l \in \{1, 2\}. \tag{5.1}$$

The nestedness condition $\mathcal{V}^{(1)} \subset \mathcal{V}^{(2)}$ implies $\Xi^{(1)} \subset \Xi^{(2)}$. The local knot insertion operators $\{\boldsymbol{R}^{e,(1,2)}\}_{e=0\dots n_e}$ can be computed from $\Xi^{(1)}$ and $\Xi^{(2)}$ by well-known knot-insertion algorithms. In particular, $\{\boldsymbol{R}^{e,(1,2)}\}$ can be constructed column-wise by the Oslo algorithm Boehm [1985]; Lyche and Morken [2008], or row-wise by the Boehm's method Boehm [1985]; Lyche and Morken [2008].

Algorithm 5.1 shows a version of the Oslo algorithm using vector operations, while Algorithm 5.2 presents its scalar version. The latter is suitable for traditional procedural programming languages. These algorithms can be combined to construct the element-local knot-insertion operators $\{\boldsymbol{R}^{e,(1,2)}\}$, as summarized in Algorithm 5.3.

Analogously, Boehm's method can be repeatedly used to produce the operators $\{\boldsymbol{R}^{e,(1,2)}\}$, as shown in Algorithm A.1. The main loop runs once through the knots of the knot vector $\Xi^{(1)}$ and the new knots to be inserted. The associated multiplicities are cached to handle the overlapping columns between operators.

Oslo's algorithm computes the knot-insertion operator's entries through a direct formula based on the coarse and fine local knots. Such an explicit formula features no data dependencies between the computation of different columns, and, therefore, Algorithm 5.3

Algorithm 5.1: `oslo1` (vector version).

Input: p: spline degree
$\boldsymbol{\Xi}^{(1)} = (\xi_1^{(1)}, \ldots, \xi_{m^{(1)}}^{1})^\top$: coarse knot vector
$\boldsymbol{\Xi}^{(2)} = (\xi_1^{(2)}, \ldots, \xi_{m^{(2)}}^{(2)})^\top$: fine knot vector
i: fine knot index such that $1 \leq i \leq m^{(2)} - p - 1$
j: coarse knot index such that $\xi_j^{(1)} \leq \xi_i^{(2)} < \xi_{i+1}^{(1)}$
Output: $\boldsymbol{b}$: vector of knot insertion coefficients

$\boldsymbol{b} = 1$
for $k = 1 \ldots p$ **do**
  $\boldsymbol{t1} = \left(\xi_{j+1-k}^{(1)}, \ldots, \xi_j^{(1)}\right)^\top$
  $\boldsymbol{t2} = \left(\xi_{j+1}^{(1)}, \ldots, \xi_{j+k}^{(1)}\right)^\top$
  $\boldsymbol{w} = \left(\xi_{i+k}^{(2)} - \boldsymbol{t1}\right) ./ (\boldsymbol{t2} - \boldsymbol{t1})$ `// ./ is the entrywise (Hadamard) division`
  $\boldsymbol{b} = \begin{bmatrix} (1 - \boldsymbol{w}) .* \boldsymbol{b} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{w} .* \boldsymbol{b} \end{bmatrix}$ `// .* is the entrywise (Hadamard) product`
end

Boehm	Oslo	bs2bs
$p(p+1)(m^{(2)} - m^{(1)})$	$\frac{1}{2}p(p+1)m^{(2)}$	$p(2p+1)s$

Table 5.1: Number of linear combinations needed to produce the element-local knot-insertion operators. s is the number of non-empty knot spans in the fine knot vector $\boldsymbol{\Xi}^{(2)}$.

is suitable for parallelization. A possible parallel version is presented in Algorithm 5.4. Instead, Algorithm A.1 inserts one knot at a time from left to right and overwrites intermediate coefficients. This iterative procedure features inherent data dependencies between computations of the same row's coefficients, but different rows could be computed in parallel. However, this is not within the scope of this work. It should also be noted that Algorithm A.1 is numerically more stable, as just convex combinations are used, while Algorithms 5.1 and 5.2 can also perform non-convex combinations.

The core operations to construct the knot-insertion operators are linear combinations of coefficients, and their count is listed in Table 5.1. The table also compares the Algorithm `bs2bs` presented in Casciola and Romani [2007]. Algorithm 5.3 (Oslo) results to be more efficient than Algorithm A.1 (Boehm) if $m^{(2)} > 2m^{(1)}$, i.e., if the number of knots (excluding the last $p+1$ repeated knots) is at least doubled from one level to the next. Therefore, Algorithm 5.3 is preferred in this work, as this condition is met in the typical case of refinement by bisection. Algorithm `bs2bs` is better than Algorithm 5.3 if the number s of non-empty spans in the fine knot vector $\boldsymbol{\Xi}^{(2)}$ is quite smaller than $m^{(2)}$. In

Algorithm 5.2: `oslo1` (scalar version).

Input: p: spline degree
$\boldsymbol{\Xi}^{(1)} = (\xi_1^{(1)}, \ldots, \xi_{m^{(1)}}^{(1)})^\top$: coarse knot vector
$\boldsymbol{\Xi}^{(2)} = (\xi_1^{(2)}, \ldots, \xi_{m^{(2)}}^{(2)})^\top$: fine knot vector
i: fine knot index, such that $1 \leq i \leq m^{(2)} - p - 1$
j: coarse knot index, such that $\xi_j^{(1)} \leq \xi_i^{(2)} < \xi_{i+1}^{(1)}$
Output: $\boldsymbol{b} = (b_1, \ldots, b_{p+1})^\top$: vector of knot insertion coefficients

$b_{p+1} = 1$
for $k = 1, \ldots, p$ **do**
  $w_2 = \frac{\xi_{j+1}^{(1)} - \xi_{i+k}^{(2)}}{\xi_{j+1}^{(1)} - \xi_{j+1-k}^{(1)}}$
  $b_{p-k+1} = w_2\, b_{p-k+2}$
  for $a = 2, \ldots, k$ **do**
    $w_1 = w_2$
    $w_2 = \frac{\xi_{j+a}^{(1)} - \xi_{i+k}^{(2)}}{\xi_{j+a}^{(1)} - \xi_{j-k+a}^{(2)}}$
    $b_{p-k+a} = (1 - w_1)\, b_{p-k+a} + w_2\, b_{p-k+a+1}$
  end
  $b_{p+1} = (1 - w_2)\, b_{p+1}$
end

particular, Algorithm `bs2bs` performs fewer combinations than Algorithm 5.3 if

$$2(1 + \frac{p}{p+1})s < n_2. \tag{5.2}$$

Therefore, such an algorithm can be preferable when $\boldsymbol{\Xi}^{(2)}$ has high multiplicity.

Considering the typical case of refinement by bisection, Figure 5.1 compares the number of linear operations used by the presented algorithms when bisecting 11 times the knot vector

$$\boldsymbol{\Xi} = (-1, -1, -1, -1, -0.8, 0.3, 1, 1, 1, 1). \tag{5.3}$$

The open-knot vector multiplicities indicate that it defines B-splines of degree $p = 3$. The parallel Oslo algorithm (Algorithm 5.4) is executed on four processors. The presented algorithms have the same asymptotic cost, but the Oslo algorithm gives a lower number of operations. Note that Figure 5.1b shows that the parallel Oslo algorithm distributes the number of combinations uniformly among the processors as the number of inserted knots increases.

The multi-level extraction operator $\boldsymbol{M}^e$ can be computed by composing rows of the refinement operators $\boldsymbol{R}^{e,(l_1,l_2)}$, as described in Sections 4.4, 4.5.1, and 4.6.1.

Algorithm 5.3: Element knot insertion operators using Oslo algorithm

Input: p: spline degree
$\boldsymbol{\Xi}^{(1)} = (\xi_1^{(1)}, \ldots, \xi_{m^{(1)}}^{(1)})^\top$: coarse knot vector
$\boldsymbol{\Xi}^{(2)} = (\xi_1^{(2)}, \ldots, \xi_{m^{(2)}}^{(2)})^\top$: fine knot vector
Output: n_e: number of non-empty knot spans
$\boldsymbol{R}^{e,(1,2)} = \boldsymbol{R}^e$, $e = 1, \ldots, n_e$: local element operators

```
1  j=p+1
2  i=1
3  e=1
4  while i≤ m-p-1 do
5  |  mult=1
6  |  while ξ^(2)_(i+mult) == ξ^(2)_i do
7  |  |  mult = mult + 1
8  |  end
9  |  lastj = j
10 |  while ξ^(1)_(j+1) ≤ ξ^(2)_i do
11 |  |  j = j+1
12 |  end
13 |  if e > 1 then
14 |  |  offs = j-lastj
15 |  |  R^e ( 1:p+1-offs, 1:p+1-mult ) = R^(e-1) ( 1+offs:p+1, 1+mult:p+1 )
16 |  end
17 |  for t= p+2-mult, ..., p+1 do
18 |  |  R^e (:, t) = oslo1(p, Ξ^(1), Ξ^(2), i, j )
19 |  |  i = i+1
20 |  end
21 |  e = e+1
22 end
23 n_e = e − 1
```

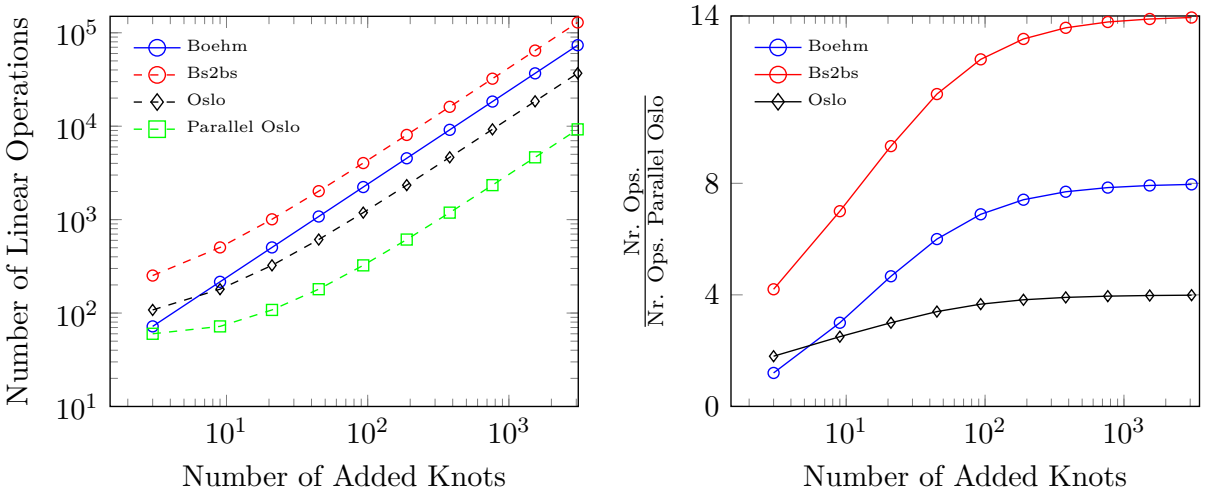
5.3 Extraction of functions

Iterative algorithms for the extraction of element basis functions can be formulated using the following basic identities. The element-local refinement operator $\boldsymbol{R}^{e,(l,\,l+1)}$ maps fine functions to coarse functions (see Equation (4.1))

$$\boldsymbol{N}^{e,(l)} = \boldsymbol{R}^{e,(l,\,l+1)}\, \boldsymbol{N}^{e,(l+1)}, \qquad l = 0 \ldots l^e - 1. \tag{5.4}$$

The operators $\{\boldsymbol{R}^{e,(l,\,l+1)}\}$ can be constructed explicitly using the algorithms presented in section 5.2. Furthermore, the standard level-l^e Bézier extractor $\boldsymbol{E}^e$ converts the Bernstein polynomials to the B-splines of level l^e, as defined in Borden et al. [2011] and analogous to Equation (2.37)

$$\boldsymbol{N}^{e,(l^e)} = \boldsymbol{E}^e\, \boldsymbol{B}^e. \tag{5.5}$$



(a) Number of linear operations

(b) Number of linear operations divided by the number of operations required by parallel Oslo

Figure 5.1: Comparison of the algorithms to construct the local knot-insertion refinement operators for 11 bisections of $\boldsymbol{\Xi} = (-1, -1, -1, -1, -0.8, 0.3, 1, 1, 1, 1)$, for $p = 3$. The parallel Oslo algorithm (Algorithm 5.4) is executed on four processors.

5.3.1 Extraction of hierarchical B-Splines

An efficient implementation of the above basic operations leverages the tensor structure of both $\boldsymbol{N}^{e,(l+1)} = \bigodot_{i=1}^{D} {}^{i}\boldsymbol{N}^{e,(l+1)}$ and the operators $\boldsymbol{R}^{e,(l,\,l+1)} = \bigodot_{i=1}^{D} {}^{i}\boldsymbol{R}^{e,(l,\,l+1)}$ and $\boldsymbol{E}^{e}$, as presented in [Borden et al., 2011]. By recursive application of the mixed-product property, Equations (2.20) and (5.4) can be written in D dimensions as [Borden et al., 2011]

$$\begin{aligned}\boldsymbol{N}^{e,(l)} &= \boldsymbol{R}^{e,(l,\,l+1)}\,\boldsymbol{N}^{e,(l+1)} \\ &= \left(\bigodot_{i=1}^{D} {}^{i}\boldsymbol{R}^{e,(l,\,l+1)}\right)\left(\bigodot_{i=1}^{D} {}^{i}\boldsymbol{N}^{e,(l+1)}\right) \\ &= \bigodot_{i=1}^{D}\left({}^{i}\boldsymbol{R}^{e,(l,\,l+1)}\;{}^{i}\boldsymbol{N}^{e,(l+1)}\right). \end{aligned} \tag{5.6}$$

This way, the computational cost of Equation (5.4) is reduced, as summarized in Table 5.2. Here, the matrices are not assumed to be sparse, as they are local to each element. Note that the table does not include the cost of Kronecker products in (5.6), as it equals the cost for computing $\boldsymbol{N}^{e,(l+1)} = \bigodot_{i=1}^{D} {}^{i}\boldsymbol{N}^{e,(l+1)}$.

Based on Equations (5.5) and (5.6), the extraction of functions can be formulated iteratively, as in Algorithm 5.5. This formulation allows using the optimized form (5.6) also in combination with hierarchical refinement. Algorithm 5.5 can be considered as the iterative version of the algorithm presented in Bressan and Mokriš [2017].

Algorithm 5.4: Element knot insertion operators (parallel

Input: p: spline degree
$\boldsymbol{\Xi}^{(1)} = (\xi_1^{(1)}, \ldots, \xi_{m^{(1)}}^{(1)})$: coarse knot vector
$\boldsymbol{\Xi}^{(2)} = (\xi_1^{(2)}, \ldots, \xi_{m^{(2)}}^{(2)})$: fine knot vector
ithread: thread index (starting from 1)
nthreads: total number of threads

Output: $\boldsymbol{R}^e$, $e = 1, \ldots, n_e$: local element operators

```
chunk = floor((m^(2)-2(p+1)+1)/nthreads)
rem = mod(m^(2)-2(p+1)+1,nthreads)
knstart = max( (ithread-1)chunk, 0 ) + min(ithread-1, rem)+1
knend = 2 p+ ithread * chunk + min(ithread, rem)+1
while knend-knstart+1 ≥ 2(p+1) and ξ^(2)_{knstart+p+1} == ξ^(2)_{knstart+p} do
    knstart = knstart+1
end
while knend-knstart+1 ≥ 2(p+1) and ξ^(2)_{knend-p-1} == ξ^(2)_{knend-p} do
    knend = knend-1
end
if knend-knstart+1 < 2(p+1) then return
Ξ = ( ξ^(2)_{knstart},...,ξ^(2)_{knend} )
cf=p+1; rf=1; e=1
while rf ≤ knend-knstart-p do
    mult=1
    while ξ^(2)_{knstart-1+rf+mult} == ξ^(2)_{knstart-1+rf} do mult = mult+1
    lastcf = cf
    while ξ^(1)_{cf+1} ≤ ξ^(2)_{knstart-1+rf} do cf = cf+1
    if e>1 then
        offs = cf-lastcf
        R^e( 1:p+1-offs, 1:p+1-mult ) = R^{e-1}( 1+offs:p+1, 1+mult:p+1 )
    end
    while ξ^(1)_{cf+1} ≤ ξ^(2)_{knstart-1+p+1} do cf = cf+1
    for t= 1, ..., p+1-mult do
        R^e(1:p+1, t) = oslo1(p, Ξ^(1), Ξ, cf, rf, ithread)
        rf=rf+1
    end
    for t= p+2-mult, ..., p+1 do
        R^e(:, t) = oslo1(p, Ξ^(1), Ξ, cf, rf)
        rf=rf+1
    end
    e=e+1
end
```

no. multiplications	construction of $\boldsymbol{R}^e$	matrix multiplication
direct evaluation of (5.4)	$\sum_{i=2}^{D}(p+1)^{2i}$	$(p+1)^{2D}$
evaluation of (5.6)	0	$D(p+1)^2$

Table 5.2: Comparison between the number of floating-point multiplications for evaluating (5.4) and (5.6).

Algorithm 5.5: `extract`. Extraction of hierarchical B-splines.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function with support on e
${}^i\boldsymbol{B}^e$: Bernstein polynomials, $i = 1 \dots D$
${}^i\boldsymbol{E}^e$: Level Bézier extractors, $i = 1 \dots D$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^{e,c}+1 \dots l^e$

Output: $\boldsymbol{H}^e$: local hierarchical functions

1 ${}^i\boldsymbol{N}^{e,(l^e)} = {}^i\boldsymbol{E}^e\ {}^i\boldsymbol{B}^e,\ i = 1 \dots D$
2 compute the level-l^e active functions as tensor product of ${}^i\boldsymbol{N}^{e,(l^e)}$ and insert them into $\boldsymbol{H}^e$
3 **for** $l = l^e, \dots, l^{e,c}+1$ **do**
4 ${}^i\boldsymbol{N}^{e,(l-1)} = {}^i\boldsymbol{R}^{e,(l)}\ {}^i\boldsymbol{N}^{e,(l)},\ i = 1 \dots D$
5 compute the level-$(l-1)$ active functions as tensor product of ${}^i\boldsymbol{N}^{e,(l-1)}$ and insert them into $\boldsymbol{H}^e$
6 **end**

5.3.2 Cost comparison

In this section, the cost of function extraction is analyzed. Following Borden et al. [2011]; Cottrell et al. [2009], shape-function routines based on Bézier extraction are considered in the form that takes a single integration point as an argument.

Table 5.3 compares the number of multiplications needed by Algorithm 5.5 to the direct extraction $\boldsymbol{C}^e\boldsymbol{B}^e$ for an explicitly constructed extraction operator $\boldsymbol{C}^e$. Here, $n_l = l^e - l^{e,c}+1$ is the number of hierarchical levels active on the element e, $l^{e,c}$ is the coarsest level of the active functions with support on element e, and n_f is the total number of functions active on element e. The cost $\texttt{cost}(\boldsymbol{C}^e)$ for constructing the extraction operator $\boldsymbol{C}^e$ is considered separately, as $\boldsymbol{C}^e$ might be cached and recomputed only when the discretization changes, although this strategy is memory demanding.

The iterative algorithm (Algorithm 5.5) performs fewer multiplications than Equation (4.8), if

$$\texttt{cost}(\boldsymbol{C}^e\boldsymbol{B}^e) > \texttt{cost}(\texttt{Algorithm 5.5}), \tag{5.7}$$

	tensor product	matrix multiplication	construction of $\boldsymbol{C}^e$
Equation (4.8)	$\sum_{i=2}^{D}(p+1)^i$ *(for* $\boldsymbol{B} = \bigodot_{i=1}^{D} {}^i\boldsymbol{B}$*)*	$n_f\,(p+1)^D$	$\texttt{cost}(\boldsymbol{C}^e)$
Algorithm 5.5 ($\mathcal{HB}$)	$n_f\,(D-1)$ *(for* $H_a = \prod_{i=1}^{D} {}^iN$*)*	$n_l\,D(p+1)^2$	0
Algorithm 5.9 ($\mathcal{THB}$)	$\sum_{i=2}^{D}(p+1)^i$ *(for* $\boldsymbol{N}^{e,(l^e)} = \bigodot_{i=1}^{D} {}^i\boldsymbol{N}^{e,(l^e)}$*)*	$(n_l-1)D(p+1)^{D+1}$ $+D(p+1)^2$	0

Table 5.3: Comparison between the number of floating-point multiplications for Equation (4.8) and algorithms 5.5 and 5.9.

with

$$\texttt{cost}(\boldsymbol{C}^e\boldsymbol{B}^e) = n_f(p+1)^D + \sum_{i=2}^{D}(p+1)^i,$$
$$\texttt{cost}(\texttt{Algorithm 5.5}) = n_l D(p+1)^2 + n_f(D-1).$$

From Inequality 5.7 it is clear that the advantage in using Algorithm 5.5 grows exponentially with D and polynomially with p, while it decays linearly with n_l (see Figure 5.2).

It is of main interest to identify when the iterative algorithm is advantageous for practical values of D and p. From Inequality 5.7, one deduces

$$\texttt{cost}(\boldsymbol{C}^e\boldsymbol{B}^e) > \texttt{cost}(\texttt{Algorithm 5.5}) \iff n_l < \phi^{\mathcal{HB}}(p, D, n_f) \tag{5.8}$$

where

$$\phi^{\mathcal{HB}}(p, D, n_f) = n_f \frac{(p+1)^D - D + 1}{D(p+1)^2} + \frac{(p+1)^{D-1} - 1}{pD}. \tag{5.9}$$

Since it must hold $n_f \geq (p+1)^D$, the following sufficient condition is obtained

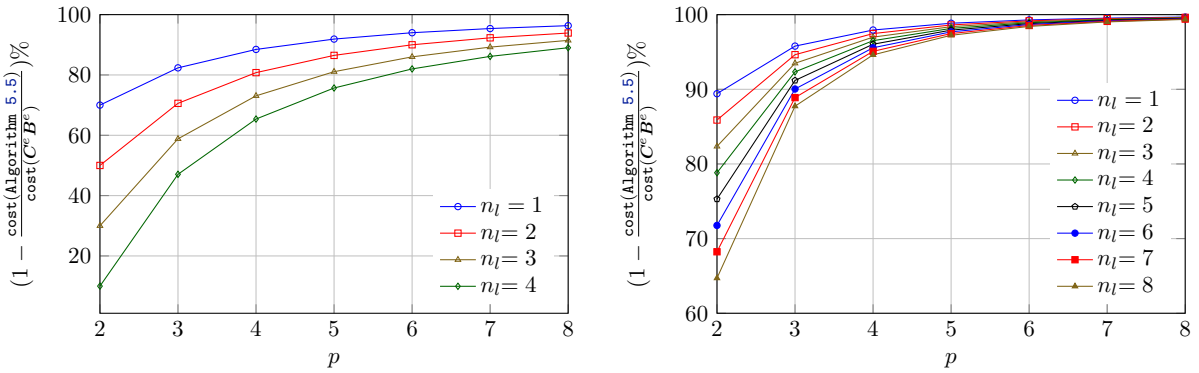
$$n_l < \phi^{\mathcal{HB}}(p, D, (p+1)^D) \implies \texttt{cost}(\boldsymbol{C}^e\boldsymbol{B}^e) > \texttt{cost}(\texttt{Algorithm 5.5}). \tag{5.10}$$

That is, the iterative algorithm is convenient, at least as long as the number n_l of active levels on every single element is not too high. Table 5.4 shows some values of $\phi^{\mathcal{HB}}$ for low D and p values, showing that in 3D, the algorithm is virtually always of practical advantage, even for low spline orders. The advantage is even more significant for higher dimensions, e.g., space-time computations. For 2D analyses and low p, $\phi^{\mathcal{HB}}$ might reach practical values of n_l. It should be noted, however, that:

- Inequality 5.10 is just a sufficient condition, but not necessary,

		D		
		2	3	4
p	2	4.5	26.3	178.8
	3	8.0	84.3	1017.3
	4	12.5	207.0	3895.3
	5	18.0	430.3	11647.8
	6	24.5	798.3	29389.8

Table 5.4: Example of values for $\phi^{\mathcal{HB}}(p, D, (p+1)^D)$.



(a) $2D$ (b) $3D$

Figure 5.2: Minimum percentage of saved multiplications. This lower bound is obtained using the pessimistic assumption $n_f = (p+1)^D$. An increase in p quickly renders Algorithm 5.5 effective also for higher n_l.

- the quantity $\texttt{cost}(\boldsymbol{C}^e)$ was ignored so far.

The assumption $n_f = (p+1)^D$ represents the worst-case scenario in which the advantage of using the iterative algorithm is lowest. However, for meaningful hierarchical refinements, the number of active functions, n_f, should grow together with the number of active levels, n_l. In case $\boldsymbol{C}^e$ is computed once in a pre-processing step and cached in memory, it must be recomputed whenever the discretization changes. Such computation increases the cost by a term of order at least $\mathcal{O}(p^{2D})$. Note that the iterative algorithm additionally spares the storage of $\boldsymbol{C}^e$.

The efficiency of the proposed algorithms can be ensured by using admissible meshes [Bracco et al., 2019; Buffa and Giannelli, 2016; Carraturo et al., 2019]. Namely the number of levels n_l containing active basis functions with support on any element is limited by a chosen parameter. The computational costs can be improved further if the shape function routine computes the basis functions at all integration points simultaneously.

5.4 Extraction of truncated hierarchical B-splines

One particular aspect of the truncated hierarchical B-splines is that each basis function is not the tensor product of univariate functions, contrarily to the hierarchical B-splines. This property inhibits the use of Equation (5.6) and Algorithm 5.5. Nevertheless, the refinement operator's tensor-product structure can still be leveraged in Equation (5.4). To this end, a suitable algorithm was proposed in Fernandes et al. [1998] and can be explained using the matrix multiplication compatibility property of Equation (2.18), as detailed in the next section.

5.4.1 Efficient multiplication by Kronecker product

Given two matrices, $\boldsymbol{X} \in \mathbb{R}^{x_1 \times x_2}$ and $\boldsymbol{Y} \in \mathbb{R}^{y_1 \times y_2}$, the product $(\boldsymbol{Y} \otimes \boldsymbol{X})\,\boldsymbol{P}$ can be written as

$$(\boldsymbol{Y} \otimes \boldsymbol{X})\,\boldsymbol{P} = (\boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})\,(\boldsymbol{I}^{y_2} \otimes \boldsymbol{X})\,\boldsymbol{P}.$$

Thanks to the block-diagonal structure of $\boldsymbol{I}^{y_2} \otimes \boldsymbol{X}$, the multiplication $(\boldsymbol{I}^{y_2} \otimes \boldsymbol{X})\,\boldsymbol{P}$ can be implemented as a sequence of standard multiplications. Moreover, if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are square matrices, the multiplications can be done in place. In particular, such multiplication takes the following block-diagonal form

$$\tilde{\boldsymbol{P}} = (\boldsymbol{I}^{y_2} \otimes \boldsymbol{X})\,\boldsymbol{P} = \begin{bmatrix} \boldsymbol{X} & & & \\ & \boldsymbol{X} & & \\ & & \ddots & \\ & & & \boldsymbol{X} \end{bmatrix} \left[\begin{array}{c} P_1 \\ \vdots \\ P_{x_2} \\ \hline P_{x_2+1} \\ \vdots \\ P_{2x_2} \\ \hline \vdots \\ \hline P_{(y_2-1)x_2+1} \\ \vdots \\ P_{y_2 x_2} \end{array}\right].$$

Such operation can be implemented as a sequence of standard multiplications by the univariate matrix $\boldsymbol{X}$ as

$$\begin{aligned} \tilde{\boldsymbol{P}}(1 : x_1, :) &= \boldsymbol{X}\,\boldsymbol{P}(1 : x_2, :), \\ \tilde{\boldsymbol{P}}(x_1 + 1 : 2x_1, :) &= \boldsymbol{X}\,\boldsymbol{P}(x_2 + 1 : 2x_2, :), \\ &\vdots \\ \tilde{\boldsymbol{P}}\left((y_2 - 1)x_1 + 1 : y_2 x_1, :\right) &= \boldsymbol{X}\,\boldsymbol{P}((y_2 - 1)x_2 + 1 : y_2 x_2, :). \end{aligned}$$

Here a MATLAB-like [MATLAB, 2019] syntax is used, where $\boldsymbol{P}(a : b, :)$ indicates a $\boldsymbol{P}$ sub-matrix composed of rows $a, a + 1, \ldots, b$ and all the columns of $\boldsymbol{P}$. The term $(\boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})$

scatters the entries of $\boldsymbol{Y}$, and its effect can be implemented as a sequence of strided multiplications

$$\hat{\boldsymbol{P}} = (\boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})\, \tilde{\boldsymbol{P}}$$

$$= \begin{bmatrix} Y_{11} & & & & Y_{12} & & & & & Y_{1y_2} & & & \\ & Y_{11} & & & & Y_{12} & & & & & Y_{1y_2} & & \\ & & \ddots & & & & \ddots & & \cdots & & & \ddots & \\ & & & Y_{11} & & & & Y_{12} & & & & & Y_{1y_2} \\ Y_{21} & & & & Y_{22} & & & & & Y_{2y_2} & & & \\ & Y_{21} & & & & Y_{22} & & & & & Y_{2y_2} & & \\ & & \ddots & & & & \ddots & & \cdots & & & \ddots & \\ & & & Y_{21} & & & & Y_{22} & & & & & Y_{2y_2} \\ \vdots & & & & \vdots & & & & & \vdots & & & \\ Y_{y_1 1} & & & & Y_{y_1 2} & & & & & Y_{y_1 y_2} & & & \\ & Y_{y_1 1} & & & & Y_{y_1 2} & & & & & Y_{y_1 y_2} & & \\ & & \ddots & & & & \ddots & & \cdots & & & \ddots & \\ & & & Y_{y_1 1} & & & & Y_{y_1 2} & & & & & Y_{y_1 y_2} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{x_1} \\ \hline P_{x_1+1} \\ P_{x_1+2} \\ \vdots \\ P_{2x_1} \\ \hline \vdots \\ \hline P_{(y_2-1)x_1+1} \\ P_{(y_2-1)x_1+2} \\ \vdots \\ P_{y_2 x_1} \end{bmatrix}.$$

Such operation can be implemented as the following sequence of strided univariate multiplications

$$\begin{aligned} \hat{\boldsymbol{P}}(1 : x_1 : (y_1-1)x_1+1, :) &= \boldsymbol{Y}\, \tilde{\boldsymbol{P}}(1 : x_1 : (y_2-1)x_1+1, :), \\ \hat{\boldsymbol{P}}(2 : x_1 : (y_1-1)x_1+2, :) &= \boldsymbol{Y}\, \tilde{\boldsymbol{P}}(2 : x_1 : (y_2-1)x_1+2, :), \\ &\vdots \\ \hat{\boldsymbol{P}}(x_1 : x_1 : y_1 x_1, :) &= \boldsymbol{Y}\, \tilde{\boldsymbol{P}}(x_1 : x_1 : y_2 x_1, :), \end{aligned}$$

where $\boldsymbol{P}(a : s : b, :)$ indicates a $\boldsymbol{P}$ sub-matrix composed of rows $a, a+s, a+2s, ..., b$ and all the columns of $\boldsymbol{P}$. The above consideration can be generalized to compute the multiplication by a Kronecker product of an arbitrary number of matrices, as published in [Fernandes et al., 1998] for square matrices ${}^{i}\boldsymbol{R} \in \mathbb{R}^{n_i \times n_i}$, $i = 1 \dots D$. A formatted version with adjusted notation is shown in Algorithm 5.6. All matrices are assumed to be dense, requiring the memory to store

$$n_1^2 + ... + n_D^2 + 2(n_1 n_2 ... n_D)$$

values. An alternative algorithm and a generalization to rectangular matrices are given in the following section.

Alternative algorithm

The term $\boldsymbol{I}^{y_2} \otimes \boldsymbol{X}$ can be implemented as a sequence of multiplications with contiguous memory access. Instead, the term $(\boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})$ that induces a scattering of $\boldsymbol{Y}$ elements and, consequently, a strided memory access. The latter multiplication can be improved utilizing the pseudo-commutativity property (2.19). The multiplication by a Kronecker product can be written as a sequence of multiplications in the form $\boldsymbol{I} \otimes \boldsymbol{X}$ followed by a perfect shuffle.

Algorithm 5.6: `kron_mult`. Kronecker multiplication algorithm [Fernandes et al., 1998].

Input: ${}^{i}\boldsymbol{R} \in \mathbb{R}^{n_i \times n_i}, i = 1 \dots D$
$\boldsymbol{P} \in \mathbb{R}^{n_1\, n_2\, \dots\, n_D \times 1}$
Output: $\boldsymbol{Q} = \left({}^{D}\boldsymbol{R} \otimes \dots \otimes {}^{2}\boldsymbol{R} \otimes {}^{1}\boldsymbol{R}\right)\, \boldsymbol{P}$.

```
nleft = n_2 n_3 ... n_D
nright = 1
for i = 1...D do
    base = 0
    for k = 1... nleft do
        for j = 1... nright do
            start = base+j
            end = start+nright (n_i − 1)
            P( start : nright : end, : ) = iR P( start : nright : end, : )
        end
        base = base + n_i  nright
    end
    nleft = nleft / n_{min{i+1,D}}
    nright = nright  n_i
end
Q = P
```

For example, in two dimensions, it holds

$$\begin{aligned}(\boldsymbol{Y} \otimes \boldsymbol{X})\, \boldsymbol{P} &= (\boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})\, (\boldsymbol{I}^{y_2} \otimes \boldsymbol{X})\, \boldsymbol{P} \\ &= \boldsymbol{K}^{(y_1,\, x_1)}\, (\boldsymbol{I}^{x_1} \otimes \boldsymbol{Y})\, \boldsymbol{K}^{(x_2,\, y_2)}\, (\boldsymbol{I}^{y_2} \otimes \boldsymbol{X})\, \boldsymbol{P}.\end{aligned}$$

Similarly, in three-dimensions one obtains

$$\begin{aligned}(\boldsymbol{Z} \otimes \boldsymbol{Y} \otimes \boldsymbol{X})\, \boldsymbol{P} &= (\boldsymbol{Z} \otimes \boldsymbol{Y} \otimes \boldsymbol{I}^{x_1})\, (\boldsymbol{I}^{z_2 y_2} \otimes \boldsymbol{X})\, \boldsymbol{P} \\ &\vdots \\ &= \boldsymbol{K}^{(z_1,\, x_1 y_1)}\, (\boldsymbol{I}^{x_1 y_1} \otimes \boldsymbol{Z}) \\ &\quad\ \boldsymbol{K}^{(y_1,\, x_1 z_2)}\, (\boldsymbol{I}^{x_1 z_2} \otimes \boldsymbol{Y}) \\ &\quad\ \boldsymbol{K}^{(x_1,\, y_2 z_2)}\, (\boldsymbol{I}^{y_2 z_2} \otimes \boldsymbol{X})\, \boldsymbol{P}.\end{aligned}$$

A generalization to an arbitrary number of dimensions is given in Algorithm 5.7. The proposed algorithm differs from Algorithm 5.6 in that it is more compact, and the strided access in the rows of $\boldsymbol{P}$ is removed. Algorithm 5.7 can improve the cache efficiency and vectorization, and enable the use of standard matrix multiplication libraries. For completeness, a generalization to rectangular matrices is given in Algorithm 5.8. In this case, the multiplication by the Kronecker product of the matrices $\{{}^{i}\boldsymbol{R} \in \mathbb{R}^{r_i \times c_i}\}_{i=1 \dots D}$ via

Algorithm 5.7: `kron_mult`. Kronecker multiplication algorithm with contiguous memory access in $\boldsymbol{P}$.

Input: ${}^{i}\boldsymbol{R} \in \mathbb{R}^{n_i \times n_i}, i = 1 \dots D$
$\boldsymbol{P} \in \mathbb{R}^{n_1\, n_2 \,\dots\, n_D \times 1}$
Output: $\boldsymbol{Q} = \left({}^{D}\boldsymbol{R} \otimes \dots \otimes {}^{2}\boldsymbol{R} \otimes {}^{1}\boldsymbol{R}\right) \boldsymbol{P}$.

```
size = n_1 n_2 ... n_D
for i = 1...D do
    stride = size / n_i
    for j = 0... stride-1 do
        T( i : stride : size, : ) = iR P( j n_i + 1 : (j + 1) n_i, : )
    end
    swap( T, P )
end
Q = P
```

no. multiplications	construction of $\left({}^{D}\boldsymbol{R} \otimes \dots \otimes {}^{2}\boldsymbol{R} \otimes {}^{1}\boldsymbol{R}\right)$	matrix multiplication
full matrix	$\sum_{i=2}^{D} (p+1)^{2i}$	$(p+1)^{2D}$
Algorithm 5.6	0	$D(p+1)^{D+1}$
Algorithm 5.7	0	$D(p+1)^{D+1}$

Table 5.5: Comparison between the number of floating-point multiplications for the direct full matrix multiplication and Algorithms 5.6 and 5.7 assuming ${}^{i}\boldsymbol{R} \in \mathbb{R}^{(p+1)\times(p+1)}, i = 1 \dots D$.

Algorithm 5.8 costs

$$\sum_{d=1}^{D} \left(\prod_{i=1}^{d} r_i \right) \left(\prod_{j=d}^{D} c_j \right).$$

If the matrices have the same number n_r of rows and the same number n_c of columns in each parametric direction, i.e., $\{{}^{i}\boldsymbol{R} \in \mathbb{R}^{n_r \times n_c}\}_{i=1\dots D}$, Algorithm 5.8 costs

$$n_r n_c^D + n_r^2 n_c^{D-1} + \dots + n_r^D n_c.$$

Cost comparison

Table 5.5 compares the costs of the Kronecker matrix's construction and multiplication to the cost of the equivalent operation performed through Algorithms 5.6 and 5.7. Both iterative algorithms present the same number of multiplications, which is lower than the standard multiplication. Moreover, these algorithms entirely avoid the cost of an explicit Kronecker product.

Algorithm 5.8: `kron_mult`. Kronecker multiplication algorithm for rectangular matrices with contiguous memory access in $\boldsymbol{P}$.

Input: ${}^{i}\boldsymbol{R} \in \mathbb{R}^{r_i \times c_i}, i = 1 \dots D$
$\boldsymbol{P} \in \mathbb{R}^{c_1\, c_2 \dots c_D \times 1}$
Output: $\boldsymbol{Q} = \left({}^{D}\boldsymbol{R} \otimes \dots \otimes {}^{2}\boldsymbol{R} \otimes {}^{1}\boldsymbol{R}\right) \boldsymbol{P}$.

```
stride = c_2 c_3 ... c_D
for i = 1...D do
    size = stride r_i
    for j = 0... stride-1 do
        T( i : stride : size, : ) = {}^iR P( j c_i + 1 : (j+1) c_i, : )
    end
    swap( T, P )
    stride = size / c_{min{i+1,D}}
end
Q = P
```

Despite Algorithms 5.6 and 5.7 performing the same number of multiplications, they differ in the order of computations and memory access. A run-time comparison is plotted in Figure 5.3, where two different implementations are compared for some values of D and p. It can be observed that the considered implementation of Algorithm 5.7 almost always performs better than the implementation of Algorithm 5.6. However, these results can differ for different compilers and hardware.

5.4.2 Extraction of truncated hierarchical B-splines

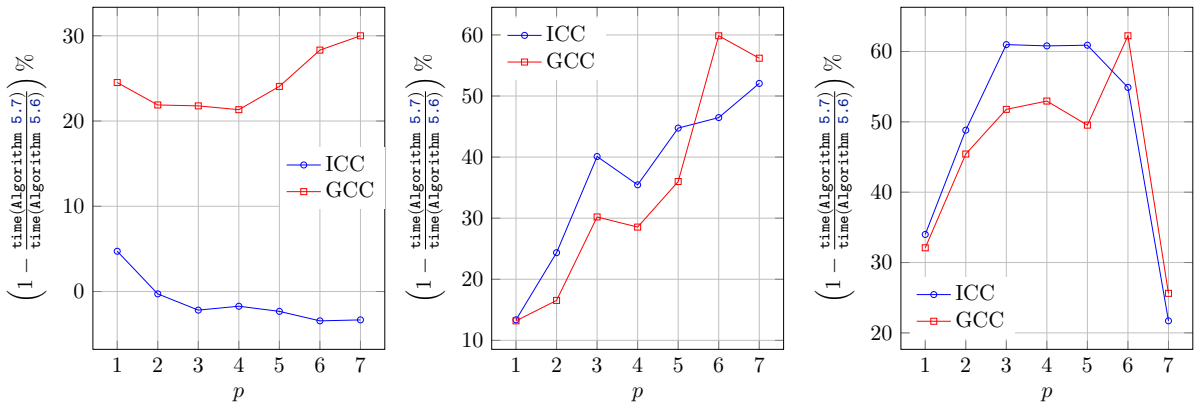
The procedure to multiply by a Kronecker matrix described in Algorithm 5.7 can be used for the iterative extraction of the truncated hierarchical B-splines, as shown in Algorithm 5.9. Here, the same notation as in Giannelli et al. [2016] is used, where $\boldsymbol{X}^{e,(l)}$ denotes a diagonal matrix with ones corresponding to the level-l element active functions of and zeros corresponding to non-active functions. Namely,

$$\boldsymbol{X}^{e,(l)} = \texttt{diag}(x_{\boldsymbol{i}}^{(l)})_{\boldsymbol{i} \in \mathcal{I}^{(l)}}, \tag{5.11}$$

where

$$x_{\boldsymbol{i}}^{(l)} = \begin{cases} 1 & \text{if } \boldsymbol{i} \in \mathcal{I}_*^{(l)}, \\ 0 & \text{otherwise,} \end{cases}$$

and $\mathcal{I}^{(l)}$ is the set of multi-indices $\boldsymbol{i}$ of all level-l functions with support on element e, while $\mathcal{I}_*^{(l)} \subset \mathcal{I}^{(l)}$ is the subset of indices corresponding to the element active-functions. Note how the truncation is not performed on the columns of the tensor-product operator $\bigodot_{i=1}^{D} {}^{i}\boldsymbol{R}^{e,(l)}$, but its effect is obtained by setting to zero the active functions in $\boldsymbol{N}^{e,(l)}$. This operation is represented by the term $(\boldsymbol{I} - \boldsymbol{X}^{e,(l)})\, \boldsymbol{N}^{e,(l)}$ and corresponds to the definition of truncation in Section 3.4.



(a) $2D$ (b) $3D$ (c) $4D$

Figure 5.3: Run-time comparison of Algorithms 5.6 and 5.7. The processes were pinned to a single core of a `Intel(R) Xeon(R) CPU E5` processor with a fixed frequency of 2.90 GHz. The values are obtained as average run-time of 10^6 computations. The compilation was performed through the commands `icc -xHost -O3 -ansi-alias` (`icc` version 15.0.0) and `g++ -ffast-math -O3 -march=native -funroll-loops` (`g++` version 4.9.0).

A similar algorithm starting from the tensor-product Bernstein polynomials is given in Algorithm 5.10. This procedure will be used in Chapter 8.

5.4.3 Cost comparison

The number of multiplications in the procedure to extract the truncated hierarchical B-splines, Algorithm 5.9, is summarized in Table 5.3. The condition for Algorithm 5.9 to perform fewer multiplications than the direct approach of explicitly constructing the extraction operator (as in Equation (4.8)) is

$$n_l < \phi^{\mathcal{THB}}(p, D, n_f) = 1 + \frac{n_f(p+1)^D - D(p+1)^2}{D(p+1)^{D+1}}. \tag{5.12}$$

Therefore, the following sufficient condition holds

$$n_l < \phi^{\mathcal{THB}}\left(p, D, (p+1)^D\right) \implies \texttt{cost}(\boldsymbol{C}^e\boldsymbol{B}^e) > \texttt{cost}(\texttt{Algorithm 5.9}), \tag{5.13}$$

with the assumption $n_f = (p+1)^D$ and neglecting $\texttt{cost}(\boldsymbol{C}^e)$. Table 5.6 shows some exemplary values for $\phi^{\mathcal{THB}}$, indicating an upper bound for n_l significantly lower than the bound obtained for hierarchical B-splines. Nevertheless, the algorithm can be useful for practical p, D, and n_l values. As in the case of hierarchical B-splines, note that the estimates are pessimistic, the iterative algorithms additionally spares the explicit computation and storage of $\boldsymbol{C}^e$, and n_l can be kept bounded by using admissible meshes [Bracco et al., 2019; Buffa and Giannelli, 2016; Carraturo et al., 2019].

5.5 Projection of DOFs for refinement

During mesh adaption in a non-linear transient analysis, it is necessary to project the solution from one mesh to another. In case the refined solution space includes the coarse

Algorithm 5.9: `extract_thb`. Extraction of truncated hierarchical B-splines.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function with support on e
${}^i\boldsymbol{B}^e$: Bernstein polynomials, $i = 1 \dots D$
${}^i\boldsymbol{E}^e$: Level Bézier extractors, $i = 1 \dots D$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c} + 1$
$\boldsymbol{X}^{e,(l)}$: Active function matrix
Output: $\boldsymbol{H}^e$: local hierarchical functions

1 ${}^i\boldsymbol{N}^{e,(l^e)} = {}^i\boldsymbol{E}^e \; {}^i\boldsymbol{B}^e, \;\; i = 1 \dots D$
2 $\boldsymbol{N}^{e,(l^e)} = \bigodot_{i=1}^{D} {}^i\boldsymbol{N}^{e,(l^e)}$
3 insert active functions from $\boldsymbol{N}^{e,(l^e)}$ into $\boldsymbol{H}^e$
4 **for** $l = l^e, \dots, l^{e,c} + 1$ **do**
5 $\quad \boldsymbol{N}^{e,(l-1)} = \texttt{kron\_mult}\left(\left\{{}^i\boldsymbol{R}^{e,(l)}\right\}, \left(\boldsymbol{I} - \boldsymbol{X}^{e,(l)}\right)\boldsymbol{N}^{e,(l)}\right)$
6 $\quad$ insert active functions from $\boldsymbol{N}^{e,(l-1)}$ into $\boldsymbol{H}^e$
7 **end**

		D		
		2	3	4
p	2	2.17	3.89	7.71
	3	2.75	6.27	16.98
	4	3.30	9.29	32.24
	5	3.83	12.97	54.99
	6	4.36	17.31	86.74

Table 5.6: Example of values for $\phi^{\mathcal{THB}}(p, D, (p+1)^D)$.

one, it is possible to express exactly the old solution in terms of the new basis. In our case, this is done through the relations dual to Equations (5.4) and (5.5)

$$\boldsymbol{P}^{e,(l+1)} = \left(\boldsymbol{R}^{e,(l,\,l+1)}\right)^\top \boldsymbol{P}^{e,(l)}, \qquad l = 0 \dots l^e - 1, \tag{5.14}$$

$$\boldsymbol{Q}^e = \left(\boldsymbol{E}^e\right)^\top \boldsymbol{P}^{e,(l^e)}. \tag{5.15}$$

The control points $\boldsymbol{P}^{e,(l+1)}$ are the $\boldsymbol{N}^{e,(l+1)}$-representation of the spline defined by the coefficients $\boldsymbol{P}^{e,(l)}$ in the basis $\boldsymbol{N}^{e,(l)}$. Similarly, $\boldsymbol{Q}^e$ denotes the Bernstein basis' control points representing the spline defined by the coefficients $\boldsymbol{P}^{e,(l^e)}$ in the basis $\boldsymbol{N}^{e,(l^e)}$. In particular,

$$\left(\boldsymbol{P}^{e,(l+1)}\right)^\top \boldsymbol{N}^{e,(l+1)} = \left(\boldsymbol{P}^{e,(l)}\right)^\top \boldsymbol{N}^{e,(l)} \tag{5.16}$$

$$\left(\boldsymbol{Q}^e\right)^\top \boldsymbol{B}^e = \left(\boldsymbol{N}^{e,(l^e)}\right)^\top \boldsymbol{P}^{e,(l^e)}. \tag{5.17}$$

See also Borden et al. [2011]; Lyche and Morken [2008]; Piegl and Tiller [1995], among others.

Algorithm 5.10: `extract_thb_tp`. Extraction of truncated hierarchical B-splines starting from tensor-product Bernstein polynomials.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function with support on e
$\boldsymbol{B}^e$: Tensor-product Bernstein polynomials
${}^i\boldsymbol{E}^e$: Level Bézier extractors, $i = 1 \dots D$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c} + 1$
$\boldsymbol{X}^{e,(l)}$: Active function matrix
Output: $\boldsymbol{H}^e$: local hierarchical functions

$\boldsymbol{N}^{e,(l^e)} = \texttt{kron\_mult}\left(\{{}^i\boldsymbol{E}^e\}, \boldsymbol{B}^e\right)$
insert active functions from $\boldsymbol{N}^{e,(l^e)}$ into $\boldsymbol{H}^e$
for $l = l^e, \dots, l^{e,c} + 1$ **do**
  $\boldsymbol{N}^{e,(l-1)} = \texttt{kron\_mult}\left(\{{}^i\boldsymbol{R}^{e,(l)}\}, \left(\boldsymbol{I} - \boldsymbol{X}^{e,(l)}\right)\boldsymbol{N}^{e,(l)}\right)$
  insert active functions from $\boldsymbol{N}^{e,(l-1)}$ into $\boldsymbol{H}^e$
end

Equations (5.14) and (5.15) multiply Kronecker operators to matrices that do not possess a tensor-product structure. To exploit Algorithm 5.7's efficiency, the projection of DOFs can be formulated as in Algorithm 5.11: a local iterative version of the algorithms in Garau and Vázquez [2018] based on Equations (5.14) and (5.15). In the projection algorithms, the element control points $\boldsymbol{P}^{e,\text{coarse}}$ to be projected are subdivided into levels. Specifically, let $\{N_i^{e,(l)}\}_{i=1,\dots,n_f^{e,(l)}}$ be the level-l functions with support on the element Ω^e, and let $\{\tilde{N}_k^{e,\text{coarse},(l)}\}_{k=1,\dots,n_a^{e,(l)}} \subset \{N_i^{e,(l)}\}_i$ be the subset of active functions on the coarse mesh. Moreover, let $a(k)$ be the association from active indices to level indices. Namely,

$$\tilde{N}_k^{e,\text{coarse},(l)} = N_{a(k)}^{e,(l)}, \qquad k = 1, \dots, n_a^{e,(l)}.$$

Let $\tilde{\boldsymbol{P}}_k^{e,\text{coarse},(l)}$ denote the control point in $\boldsymbol{P}^{e,\text{coarse}}$ associated with the active function $\tilde{N}_k^{e,\text{coarse},(l)}$. The control points $\boldsymbol{P}^{e,\text{coarse},(l)} \in \mathbb{R}^{n_f^{e,(l)} \times m_f}$ in Algorithm 5.11 are defined as

$$P_{ij}^{e,\text{coarse},(l)} = \begin{cases} \left[\tilde{\boldsymbol{P}}_k^{e,\text{coarse},(l)}\right]_j & \text{if } i = a(k) \text{ for some } 1 \le k \le n_a^{e,(l)}, \\ 0 & \text{otherwise.} \end{cases}$$

The symbols $\boldsymbol{X}^{\text{coarse},e,(l)}$ and $\boldsymbol{X}^{\text{fine},e,(l)}$ denote the active function matrices on the coarse and fine mesh, respectively, as defined in Equation (5.11).

Algorithm 5.11 coincides with the spline-evaluation algorithm given in Giannelli et al. [2016] when projecting the degrees of freedom to the finest level (see Algorithm 5.12).

5.6 Projection of DOFs for mesh coarsening

During mesh coarsening, the fine-mesh solution is not representable exactly on the coarse mesh, but a suitable projection of the DOFs can retain good approximations. To perform a

Algorithm 5.11: `coarse_to_fine`. DOFs projection algorithm.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function of coarse mesh with support on e
$\boldsymbol{P}^{\text{coarse},e,(l)}$: Coarse mesh control points, $l = l^e \dots l^{e,c}+1$
$\boldsymbol{X}^{\text{coarse},e,(l)}$: Coarse mesh active function matrix, $l = l^e \dots l^{e,c}+1$
$\boldsymbol{X}^{\text{fine},e,(l)}$: Fine mesh active function matrix, $l = l^e \dots l^{e,c}+1$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c}+1$
Output: $\boldsymbol{P}^{\text{fine},e,(l)}$: Fine control points, $l = l^e \dots l^{e,c}+1$

$\boldsymbol{P}^e = \boldsymbol{P}^{\text{coarse},e,(l^{e,c})}$

$\boldsymbol{P}^{\text{fine},e,(l^{e,c})} = \boldsymbol{X}^{\text{fine},e,(l^{e,c})}\, \boldsymbol{P}^e$

for $l = l^{e,c}+1, \dots, l^e$ **do**

$(\mathcal{HB})$ $\quad \boldsymbol{P}^e = (\boldsymbol{R}^{e,(l)})^\top (\boldsymbol{I} - \boldsymbol{X}^{\text{fine},e,(l\text{-}1)})\boldsymbol{P}^e + \boldsymbol{P}^{\text{coarse},e,(l)}$

$= \texttt{kron\_mult}\left(\{({}^i\boldsymbol{R}^{e,(l)})^\top\}, (\boldsymbol{I} - \boldsymbol{X}^{\text{fine},e,(l\text{-}1)})\boldsymbol{P}^e\right) + \boldsymbol{P}^{\text{coarse},e,(l)}$

$(\mathcal{THB})$ $\quad \boldsymbol{P}^e = (\boldsymbol{I} - \boldsymbol{X}^{\text{coarse},e,(l)})(\boldsymbol{R}^{e,(l)})^\top \boldsymbol{P}^e + \boldsymbol{P}^{\text{coarse},e,(l)}$

$= (\boldsymbol{I} - \boldsymbol{X}^{\text{coarse},e,(l)})\texttt{kron\_mult}\left(\{({}^i\boldsymbol{R}^{e,(l)})^\top\}, \boldsymbol{P}^e\right) + \boldsymbol{P}^{\text{coarse},e,(l)}$

$\boldsymbol{P}^{\text{fine},e,(l)} = \boldsymbol{X}^{\text{fine},e,(l)}\, \boldsymbol{P}^e$

end

local $\mathcal{L}_2$-projection without explicitly integrating, assembling, and solving a linear system of equations, the Bézier projection [Lorenzo et al., 2017; Thomas et al., 2015] is employed. This section shows how a modification of the original formulation allows leveraging the tensor structure through the previous sections' algorithms.

5.6.1 Standard Bézier projection

Following closely Lorenzo et al. [2017]; Thomas et al. [2015], the Bézier projection is introduced utilizing the Gramian matrix $\boldsymbol{G}^p$ of the Bernstein polynomials $B_i^p(\xi)$ of degree p and index $i \in \{1, ..., p+1\}$ defined on the reference interval $[-1, 1]$

$$G_{ij}^p = \int_{-1}^{1} B_i^p(\xi)\, B_j^p(\xi)\, \mathrm{d}\xi \qquad\qquad i, j = 1, 2, ..., p+1.$$

The above integral can be computed in closed form (see [Doha et al., 2011; Thomas et al., 2015]). In D dimensions with polynomial degrees $\boldsymbol{p} = (p_1, \dots, p_D)$, $\boldsymbol{G}^{\boldsymbol{p}}$ has a tensor-product structure

$$\boldsymbol{G} = \boldsymbol{G}^{\boldsymbol{p}} = \boldsymbol{G}^{p_D} \otimes \dots \otimes \boldsymbol{G}^{p_1}. \tag{5.18}$$

Let $\boldsymbol{B}^{I,p}$ be the Bernstein polynomials of degree p defined on the interval $I = (a, b) \subset \mathbb{R}$

$$B_i^{I,p}(\xi) = \binom{p}{i-1} \frac{(b-\xi)^{p-(i-1)}(\xi - a)^{i-1}}{(b-a)^p}.$$

Algorithm 5.12: `coarse_to_finest`. Specialization of Algorithm 5.11 for projecting to the finest level.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function with support on e of the coarse mesh
$\boldsymbol{P}^{\text{coarse},e,(l)}$: Coarse mesh control points, $l = l^e \dots l^{e,c}+1$
$\boldsymbol{X}^{\text{coarse},e,(l)}$: Coarse mesh active function matrix, $l = l^e \dots l^{e,c}+1$
${}^i\boldsymbol{R}^{e,(l)}$:Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c}+1$
Output: $\boldsymbol{P}^e$: Fine control points.

1 $\boldsymbol{P}^e = \boldsymbol{P}^{\text{coarse},e,(l^{e,c})}$
2 **for** $l = l^{e,c}+1, \dots, l^e$ **do**

$$(\mathcal{HB}) \quad \boldsymbol{P}^e = (\boldsymbol{R}^{e,(l)})^\top \boldsymbol{P}^e + \boldsymbol{P}^{\text{coarse},e,(l)}$$
$$= \texttt{kron\_mult}\left(\{({}^i\boldsymbol{R}^{e,(l)})^\top\}, \boldsymbol{P}^e\right) + \boldsymbol{P}^{\text{coarse},e,(l)}$$

3

$$(\mathcal{THB}) \quad \boldsymbol{P}^e = (\boldsymbol{I} - \boldsymbol{X}^{\text{coarse},e,(l)})(\boldsymbol{R}^{e,(l)})^\top \boldsymbol{P}^e + \boldsymbol{P}^{\text{coarse},e,(l)}$$
$$= (\boldsymbol{I} - \boldsymbol{X}^{\text{coarse},e,(l)})\texttt{kron\_mult}\left(\{({}^i\boldsymbol{R}^{(l)})^\top\}, \boldsymbol{P}^e\right) + \boldsymbol{P}^{\text{coarse},e,(l)}$$

4 **end**

Let $\boldsymbol{A}^{(J,I)}$ be the linear operator such that [Farouki and Neff, 1990; Thomas et al., 2015]

$$\boldsymbol{B}^{I,p} = (\boldsymbol{A}^{(J,I)})^\top \boldsymbol{B}^{J,p}. \tag{5.19}$$

Namely, $\boldsymbol{A}^{(J,I)}$ is the operator expressing the Bernstein polynomials $\boldsymbol{B}^{I,p}$ defined on the interval I as linear combination of Bernstein polynomials $\boldsymbol{B}^{J,p}$ defined on another interval $J = (\tilde{a}, \tilde{b}) \subset \mathbb{R}$. The entries of such matrix can be computed according to the formula [Farouki and Neff, 1990; Thomas et al., 2015]

$$A^{(J,I)}_{jk} = \sum_{i=\max\{1,j+k-p+1\}}^{\min\{j,k\}} B^{I,j\text{-}1}_i(\tilde{b})\; B^{I,p\text{-}j\text{-}1}_{k\text{-}i}(\tilde{a}) \qquad j,k = 1,2,...,p+1.$$

Algorithm 5.13: `coarse_to_Bernstein`. Specialization of Algorithm 5.11 for projecting to finest level Bernstein representation.

Input: l^e: level of element e
$l^{e,c}$: level of coarsest active function of coarse mesh with support on e
$\boldsymbol{P}^{\text{coarse},e,(l)}$: Coarse mesh control points, $l = l^e \dots l^{e,c}+1$
$\boldsymbol{X}^{\text{coarse},e,(l)}$: Coarse mesh active function matrix,$l = l^e \dots l^{e,c}+1$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c}+1$
${}^i\boldsymbol{E}^e$: Level Bézier extraction operators, $i = 1 \dots D$
Output: $\boldsymbol{Q}^e$: Bernstein control points.

1 $\boldsymbol{P}^e = \texttt{coarse\_to\_finest}(l^e, l^{e,c}, \boldsymbol{P}^{\text{coarse},e,(l)}, \boldsymbol{X}^{\text{coarse},e,(l)}, \{{}^i\boldsymbol{R}^{e,(l)}\})$
2 $\boldsymbol{Q}^e = (\boldsymbol{E}^e)^\top \boldsymbol{P}^e = \texttt{kron\_mult}\left(\{({}^i\boldsymbol{E}^e)^\top\}, \boldsymbol{P}^e\right)$

Similarly to $\boldsymbol{G}^{\boldsymbol{p}}$, $\boldsymbol{A}^{(J,I)}$ has a tensor-product structure $\boldsymbol{A}^{(J,I)} = \boldsymbol{A}^{(J_{i_D},I_{i_D})} \otimes \ldots \otimes \boldsymbol{A}^{(J_{i_1},I_{i_1})}$, for the Cartesian-product intervals $I = I_{i_1} \times ... \times I_{i_D}$ and $J = J_{i_1} \times ... \times J_{i_D}$.

Finally, let $\boldsymbol{L}^e$ be the transmission refinement operator introduced in [Lorenzo et al., 2017]. Such operator can be obtained by knot insertion techniques consistent with the global knot insertion operation such that it satisfies the relation

$$\boldsymbol{N}^{e,(l^{e,c})} = \boldsymbol{L}^e \boldsymbol{H}^e \tag{5.20}$$

$$= \boldsymbol{L}^e \boldsymbol{C}^e \boldsymbol{B}^e. \tag{5.21}$$

Such operator maps the hierarchical functions $\boldsymbol{H}^e$ to the coarse single-level functions $\boldsymbol{N}^{e,(l^{e,c})}$. Its effect can be obtained by an iterative algorithm similar to Algorithm 5.9. See Lorenzo et al. [2017] for further details.

The local $\mathcal{L}_2$-projection from the source (fine) elements $\{\Omega^{e_j}\}_{j=1...n_c}$ to the parent (coarse) element Ω^e can be computed explicitly through the Bézier formulation [Lorenzo et al., 2017]

$$\begin{aligned}
\boldsymbol{Q}^{e_j} &= (\boldsymbol{C}^{e_j})^\top \boldsymbol{P}^{e_j} && \text{(Bézier control values on } \Omega^{e_j}\text{)},\\
\boldsymbol{Q}^{e} &= \sum_{j=1}^{n_c} \frac{\texttt{vol}(\Omega^{e_j})}{\texttt{vol}(\Omega^{e})}\, \boldsymbol{G} (\boldsymbol{A}^{(e,e_j)})^{-\top} \boldsymbol{G}\, \boldsymbol{Q}^{e_j} && \text{(Bézier control values on } \Omega^{e}\text{)},\\
\boldsymbol{P}^{e} &= (\boldsymbol{L}^{e})^\top \left(\boldsymbol{L}^e \boldsymbol{C}^e\right)^{-\top} \boldsymbol{Q}^e && \text{(spline control values on } \Omega^{e}\text{)},
\end{aligned} \tag{5.22}$$

where $(\boldsymbol{A}^{(e,e_j)})^{-\top}$ maps the Bernstein polynomials defined on the element-e_j local coordinates to the Bernstein polynomials defined on the local coordinates of the coarse element e (cf. Lorenzo et al. [2017]; Thomas et al. [2015]).

Equation (5.22) gives an expression for the fine control points' projection to the coarse element e. Projections performed on neighboring elements will yield different projected local values for shared global control points. The global control value $\boldsymbol{P}_A$ of the Ath hierarchical function, H_A, is obtained by averaging the element-local control point values $\boldsymbol{P}^{e_k}(H_A)$ produced by the Bézier projection local to each element Ω^{e_k} contained in the support of H_A.

In particular, let $E(H_A)$ be the set of elements indices such that for any $e_k \in E(H_A)$, element Ω^{e_k} is contained in the support of H_A, i.e., $\Omega^e \cap supp(H_A) \neq \emptyset$. The final global projection is obtained by a weighted sum of the element control point values of elements with index in $E(H_A)$

$$\boldsymbol{P}_A = \sum_{e_k \in E(H_A)} \omega^{e_k}(H_A)\, \boldsymbol{P}^{e_k}(H_A)$$

where $\omega^{e_k}(H_A)$ is the weight

$$\omega^{e_k}(H_A) = \frac{\beta^{e_k}(H_A)}{\sum_{e'_k \in E(H_A)} \beta^{e'_k}(H_A)},$$

$$\beta^{e_k}(H_A) = \int_{\Omega^{e_k}} H_A \, \mathrm{d}\Omega.$$

In Thomas et al. [2015], it is presented an approximation to the weights $\omega^{e_k}(H_A)$ for meshes that are not hierarchically refined. In particular, when the parametric element span $\hat{\Omega}^{e_k}$ is proportionally similar to the physical element Ω^{e_k}, the weights can be approximated by computing the integrals in the parametric domain $\hat{\Omega}^{e_k}$. Such an estimate allows calculating the weights in terms of integrals of Bernstein polynomials

$$\int_a^b B_i^p(\xi)\,\mathrm{d}\xi = \frac{b-a}{p+1}. \tag{5.23}$$

These integrals can be transformed to B-spline integrals through the Ath row $C_{Aj}^{e_k}$ of the element extraction operator $\boldsymbol{C}^{e_k}$, giving the following approximation to the averaging weights

$$\begin{aligned}
\omega^{e_k}(H_A) &\approx \frac{\hat{\beta}^{e_k}(H_A)}{\sum_{e_k' \in E(H_A)} \hat{\beta}^{e_k'}(H_A)}, \\
\hat{\beta}^{e_k}(H_A) &= \gamma \int_{\hat{\Omega}^{e_k}} H_A \,\mathrm{d}\Omega && (5.24) \\
&= \mathtt{vol}(\hat{\Omega}^{e_k}) \sum_{j=1}^{p+1} C_{Aj}^{e_k}, && (5.25)
\end{aligned}$$

where

$$\gamma = \prod_{i=1}^{D} (p_i + 1) \tag{5.26}$$

See Lorenzo et al. [2017] for further details.

5.6.2 Modified Bézier projection

The Bézier projection (5.22) is based on the Kronecker matrices $\boldsymbol{A}$ and $\boldsymbol{G}$, while the operators $\boldsymbol{C}^{e_j}$, $\boldsymbol{C}^e$, and $\boldsymbol{L}^e$ do not possess the tensor-product structure. Following the previous arguments, the implementation strategy uses Algorithm 5.7 to perform the multiplications by Kronecker matrices and Algorithms 5.11 and 5.13 to compute the effect of $(\boldsymbol{C}^{e_j})^\top$ and $(\boldsymbol{L}^e)^\top$, respectively. However, these algorithms cannot be used to calculate the effect of $(\boldsymbol{L}^e\boldsymbol{C}^e)^{-\top}$.

To avoid an explicit computation, an alternative formulation of Equation (5.20) can be obtained through the operators $\boldsymbol{A}$ and $\boldsymbol{E}$. In particular, it holds

$$\begin{aligned}
\boldsymbol{N}^{e,(l^e,c)} &= \boldsymbol{E}^{e_c}\boldsymbol{B}^{e_c}, \\
&= \boldsymbol{E}^{e_c}(\boldsymbol{A}^{(e,e_c)})^\top \boldsymbol{B}^e. && (5.27)
\end{aligned}$$

The linear independence of $\boldsymbol{B}^e$ allows deducing from Equations (5.20) and (5.27) the following identity

$$\boldsymbol{L}^e\boldsymbol{C}^e = \boldsymbol{E}^{e_c}(\boldsymbol{A}^{(e,e_c)})^\top.$$

Algorithm 5.14: Modified Bézier projection algorithm.

Input: l^e, l^{e_j}: level of element e, e_j, respectively
$l^{e,c}$, $l^{e_j,c}$: level of coarsest active function with support on e, e_j respectively
$\boldsymbol{P}^{e_j}$: Control points of children in fine mesh
$\boldsymbol{X}^{\text{coarse},e,(l)}$: Coarse mesh active function matrix, $l = l^e \dots l^{e,c} + 1$
$\boldsymbol{X}^{\text{fine},e,(l)}$: Fine mesh active function matrix, $l = l^e \dots l^{e,c} + 1$
${}^i\boldsymbol{R}^{e,(l)}$: Refinement operators, $i = 1 \dots D$, $l = l^e \dots l^{e,c} + 1$
${}^i\boldsymbol{E}^e$: Level Bézier extraction operators, $i = 1 \dots D$

Output: $\boldsymbol{P}^e$: Parent element control points, $l = 0 \dots N - 1$.

1 **for each** child element e_j **do**

2 $\quad \boldsymbol{Q}^{e_j} = \texttt{coarse\_to\_Bernstein}\left(l^{e_j},\ l^{e_j,c},\ \boldsymbol{P}^{e_j},\ \boldsymbol{X}^{\text{fine},e_j,(l)},\ \{{}^i\boldsymbol{R}^{e_j,(l)}\},\ \{{}^i\boldsymbol{E}^{e_j}\} \right)$

3 **end**

4 $\boldsymbol{Q}^e = \sum_{e_j} \frac{\texttt{vol}(\Omega^{e_j})}{\texttt{vol}(\Omega^e)}\ \texttt{kron\_mult}\left(\{ ({}^i\boldsymbol{A}^{(e,e_j)})^{-\top}\ {}^i\boldsymbol{G} \},\ \{\boldsymbol{Q}^{e_j}\} \right)$

5 $\boldsymbol{P}^{e_c} = \texttt{kron\_mult}\left(\{ ({}^i\boldsymbol{E}^{e_c})^{-\top}\ {}^i\boldsymbol{A}^{(e_c,e)}\ {}^i\boldsymbol{G}^{(-1)} \},\ \boldsymbol{Q}^e \right)$

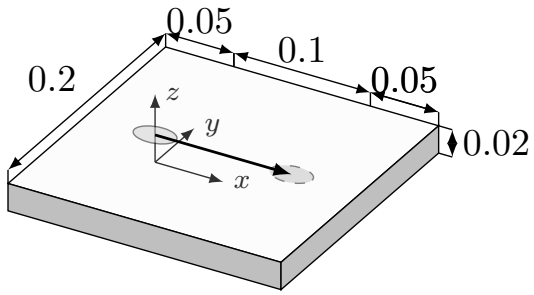
6 $\boldsymbol{P}^e = \texttt{coarse\_to\_fine}\left(l^e,\ l^{e,c},\ \boldsymbol{P}^{e_c},\ \boldsymbol{X}^{\text{coarse},e,(l)},\ \boldsymbol{X}^{\text{fine},e,(l)},\ \{{}^i\boldsymbol{R}^{e,(l)}\} \right)$

This relation allows to modify the standard Bézier projection to implement the effect of $(\boldsymbol{L}^e\boldsymbol{C}^e)^{-\top}$ without explicitly computing $\boldsymbol{L}^e$, $\boldsymbol{C}^e$, and the inverse of their product, but through Algorithm 5.7, as shown in Algorithm 5.14. Note that multiplications of small univariate matrices are performed for each parametric direction, and the result is used as input for Algorithm 5.7.

The approximated averaging quantities $\hat{\beta}^e(H_A)$ of Equation (5.25) can be generalized to hierarchically-refined meshes and computed through Algorithms 5.5 and 5.9. Indeed, for uniform polynomial degree p, the weights $\hat{\beta}^e(H_A)$ can be calculated simultaneously for all element hierarchical functions $\boldsymbol{H}^e$ as

$$\begin{aligned}
\hat{\beta}^e(\boldsymbol{H}^e) &= \gamma \int_{\hat{\Omega}^e} \boldsymbol{H}^e \, \mathrm{d}\Omega, \\
&= \gamma \int_{\hat{\Omega}^e} \boldsymbol{C}^e \boldsymbol{B}^e \, \mathrm{d}\Omega, \\
&= \gamma\, \boldsymbol{C}^e \bigodot_{i=1}^{D} \int_{{}^i\hat{\Omega}^e} {}^i\boldsymbol{B}^e \, \mathrm{d}\Omega, \\
&= \boldsymbol{C}^e \bigodot_{i=1}^{D} \texttt{vol}({}^i\hat{\Omega}^e)\ \mathbf{1},
\end{aligned}$$

where $\mathbf{1} \in \mathbb{R}^{p+1}$ is a vector of ones and ${}^i\hat{\Omega}^e$ is the interval in the ith parametric direction composing the parametric domain: $\hat{\Omega}^e = {}^1\hat{\Omega}^e \times \ldots \times {}^D\hat{\Omega}^e$. The weight $\hat{\beta}^e(H_A)$ can be computed by Algorithm 5.5 for the hierarchical B-splines and by Algorithm 5.9 for the



(a) Geometric setup $[m]$.

Specific heat (h_c)	600J/(kg °C)
Density (ρ)	7820kg/m^3
Heat conductivity (k)	29W/(m °C)
Heat power (Q)	5083W
Source speed (v)	5mm/s
Source radii r_x	15mm
Source radii r_y	10mm
Source radii r_z	2mm
Number of time steps	200
Total time	20s

(b) Parameters.

Figure 5.4: Setup of the traveling heat source example.

truncated basis

$$\hat{\beta}^{e,\mathcal{HB}}(\boldsymbol{H}^e) = \texttt{extract}(l^e,\, l^{e,c},\, \{\texttt{vol}({}^i\Omega^e)\mathbf{1}\},\, \{{}^i\boldsymbol{E}^e\},\, \{{}^i\boldsymbol{R}^{e,(l)}\}),$$
$$\hat{\beta}^{\mathcal{THB},e}(\boldsymbol{H}^e) = \texttt{extract\_thb}(l^e,\, l^{e,c},\, \{\texttt{vol}({}^i\Omega^e)\mathbf{1}\},\, \{{}^i\boldsymbol{E}^e\},\, \{{}^i\boldsymbol{R}^{e,(l)}\},\, \{\boldsymbol{X}^{e,(l)}\}).$$

5.7 Numerical Examples

The run-time of the proposed algorithms is compared with an explicit extraction-operator construction through two numerical examples. The first measures the performance of a linear transient computation with dynamic mesh refinement and coarsening, while the second shows similar results on a non-linear example with a static pre-refined mesh.

5.7.1 Traveling Heat Source

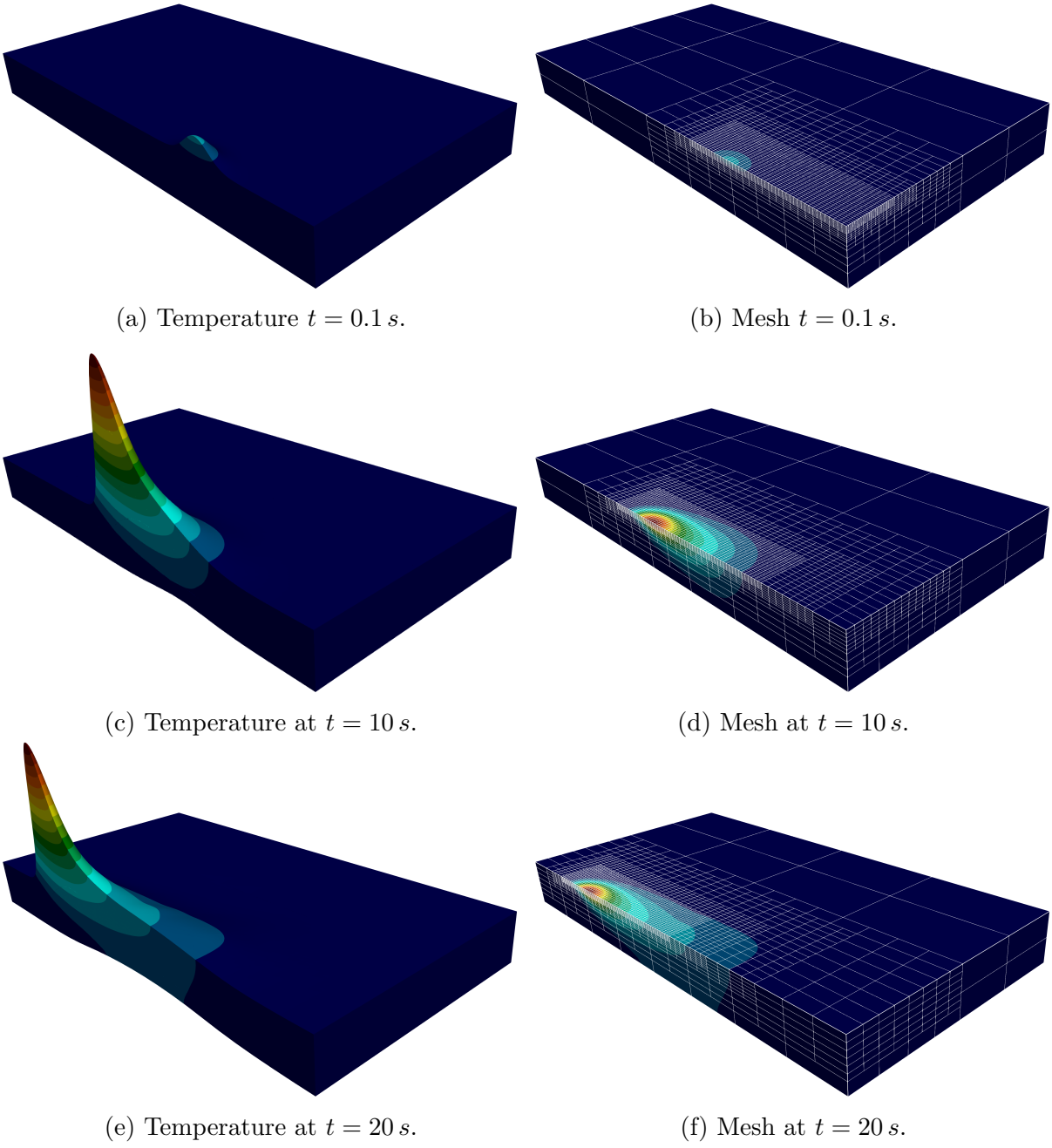
Consider the example of a moving heat source presented in Kollmannsberger et al. [2018], where a 3D ellipsoidal heat source travels at a constant speed through a rectangular plate Ω with a constant thickness (see Figure 5.4a). This process is modeled by the heat equation

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k\, \nabla T) = q, \qquad \text{on } \Omega,$$

where the (constant) material parameters are listed in Figure 5.4b, and q is defined as

$$q(x,y,z,t) = \frac{6\sqrt{3}Q}{\pi\sqrt{\pi} r_x r_y r_z} \exp\left(-3\frac{\tilde{x}^2}{r_x^2} - 3\frac{\tilde{y}^2}{r_y^2} - 3\frac{\tilde{z}^2}{r_z^2}\right), \tag{5.28}$$

$\tilde{x} = x - vt$, $\tilde{y} = y$, $\tilde{z} = z - 0.02$ being the coordinates shifted to the center of the heat source traveling with velocity v in the positive x-direction Kollmannsberger et al. [2018]. A no-flux boundary condition is applied on all faces of the plate.



(a) Temperature $t = 0.1\,s$. (b) Mesh $t = 0.1\,s$.

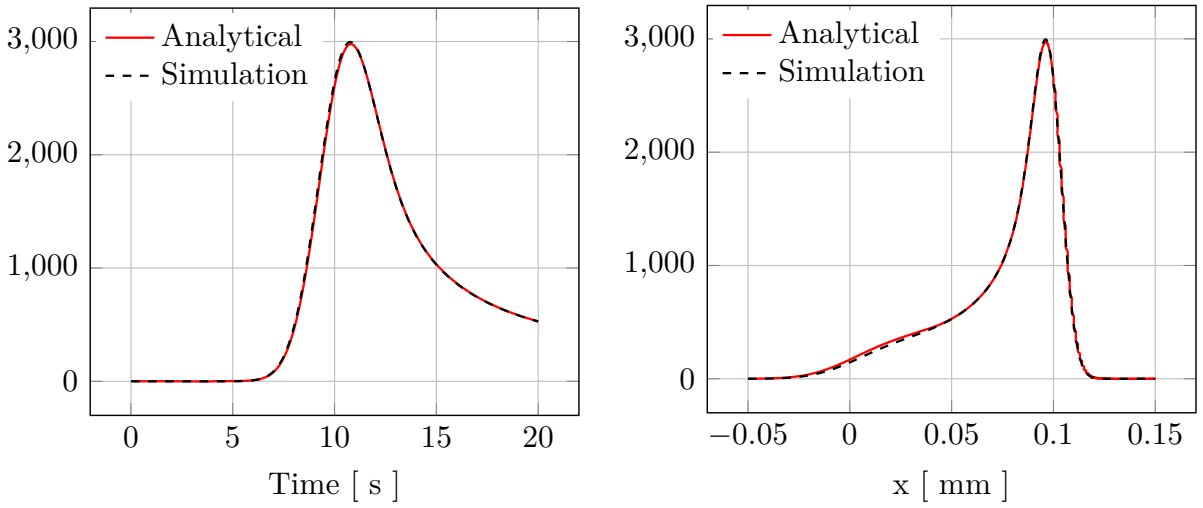
(c) Temperature at $t = 10\,s$. (d) Mesh at $t = 10\,s$.

(e) Temperature at $t = 20\,s$. (f) Mesh at $t = 20\,s$.

Figure 5.5: Solution (left) warped along z-axis (warping factor $2\,10^{-5}$) and mesh (right).

The problem is solved using the backward Euler time integration scheme on a $4 \times 4 \times 2$ grid with $p = 2$ and 5 hierarchical B-spline refinement levels around the moving heat source to capture the steep gradients. The mesh and corresponding solution are shown in Figure 5.5.

An analytical solution is available on an infinite domain [Fachinotti et al., 2011]. Assuming the boundary influence is negligible over the chosen time interval, Figure 5.6 compares the analytical solution to the obtained numerical temperature. Figure 5.7 shows that the



(a) Temperature over time at $(x, y, z) = (0.05, 0, 0.2)$.

(b) Temperature on the line $x \in [-0.05, 0.1]$, $(y, z) = (0, 0.02)$ at $t = 20\,s$.

Figure 5.6: Solution for the traveling heat source example compared with the analytical solution in space and time.

local refinement keeps the number of DOFs bounded over time.

The CPU time of a C++ implementation of Algorithm 5.5 is compared to a direct extraction $\boldsymbol{C}^e\boldsymbol{B}^e$ for an explicitly constructed extraction operator $\boldsymbol{C}^e$. The measured time includes the matrix multiplication and tensor-product operations listed in Table 5.3, including the Bernstein polynomials' evaluation. The measurements are taken on an `Intel(R) Xeon(R) CPU E5` CPU with a fixed frequency of 2.90 GHz for the first 25 time steps and $p \in \{2, 3, 4\}$. Note that the time for constructing the extraction operator $\boldsymbol{C}^e$ is ignored. The performance is evaluated according to the amount of time saved by Algorithm 5.5, calculated as

$$\texttt{time\_saved} = (1 - \frac{\texttt{time(Algorithm 5.5)}}{\texttt{time}(\boldsymbol{C}^e\boldsymbol{B}^e)})\%, \tag{5.29}$$

and the reduction factor, according to the formula

$$\texttt{reduction\_factor} = \frac{\texttt{time}(\boldsymbol{C}^e\boldsymbol{B}^e)}{\texttt{time(Algorithm 5.5)}}. \tag{5.30}$$

The results are summarized in Table 5.7.

5.7.2 Finite-strain J2 elastoplasticity

This chapter is concluded with a finite-strain elastoplastic example. Since the adopted plasticity model is standard in the literature, it is introduced as briefly as possible. For a more detailed explanation, see, e.g., de Souza Neto et al. [2009]; Igelbüscher et al. [2019]; Korelc and Stupkiewicz [2014]; Simo and Hughes [1998]; Simo and Miehe [1992]; Wriggers and Hudobivnik [2017].

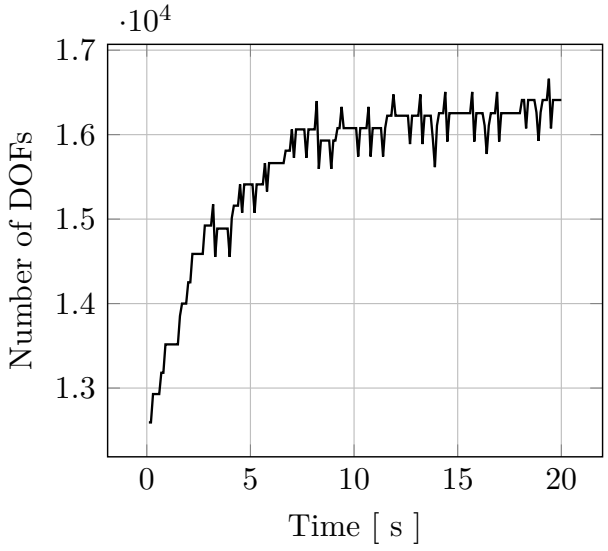


Figure 5.7: The number of DOFs is kept bounded over time by the dynamic refinement and coarsening around the moving heat source.

p	time_saved	reduction_factor
2	60.8%	2.5
3	92.5%	13
4	96.8%	31

Table 5.7: Traveling Heat Source example: time saved in extracting the functions through Algorithm 5.5 instead of direct extraction $\boldsymbol{C}^e\boldsymbol{B}^e$ for an explicitly constructed extraction operator $\boldsymbol{C}^e$.

In the following, a multiplicative decomposition of the deformation gradient $\boldsymbol{F} = \boldsymbol{F}^e\boldsymbol{F}^p$ is assumed, and the elastic deformation is characterized by an energy function

$$\Psi = \frac{\lambda}{4}\left(I_3 - 1 - \ln I_3\right) + \frac{\mu}{2}\left(I_1 - 3 - \ln I_3\right),$$

where λ and μ are the Lamé constants, and I_1 and I_3 are the first and third invariant of the elastic left Cauchy-Green tensor. The elastic region is defined by the classical von Mises yield criterion

$$f\left(\boldsymbol{\tau}, \bar{\alpha}\right) = \sqrt{\frac{3}{2}\,\mathtt{dev}[\boldsymbol{\tau}] : \mathtt{dev}[\boldsymbol{\tau}]} - K(\bar{\alpha}) \leq 0,$$

where $\mathtt{dev}[\boldsymbol{\tau}]$ is the deviatoric Kirchhoff stress, $\bar{\alpha}$ is the equivalent plastic strain, and $K(\bar{\alpha})$ is the nonlinear hardening function

$$K(\bar{\alpha}) = \sigma_0 + (\sigma_\infty - \sigma_0)(1 - e^{-\delta\bar{\alpha}}) + H\bar{\alpha}, \tag{5.31}$$

with initial stress σ_0, saturation stress σ_∞, hardening exponent δ, and linear hardening coefficient H.

Lamé's first parameter (λ)	110,743MPa
Shear modulus (μ)	80,194MPa
Initial yield strength (Y_0)	450.0MPa
Saturation strength (Y_∞)	715.0MPa
Linear hardening parameter (H)	129.24MPa
Hardening exponent (δ)	16.93

Table 5.8: Finite J2 elastoplastic material parameters as in Hubrich and Düster [2019].

The evolution of the equivalent plastic strain is governed by $\partial_t \bar{\alpha} = \gamma$ for a consistency parameter γ subject to the Kuhn-Tucker and consistency conditions

$$\gamma \geq 0, \tag{5.32}$$
$$f \leq 0, \tag{5.33}$$
$$\gamma f = \gamma\, \partial_t f = 0. \tag{5.34}$$

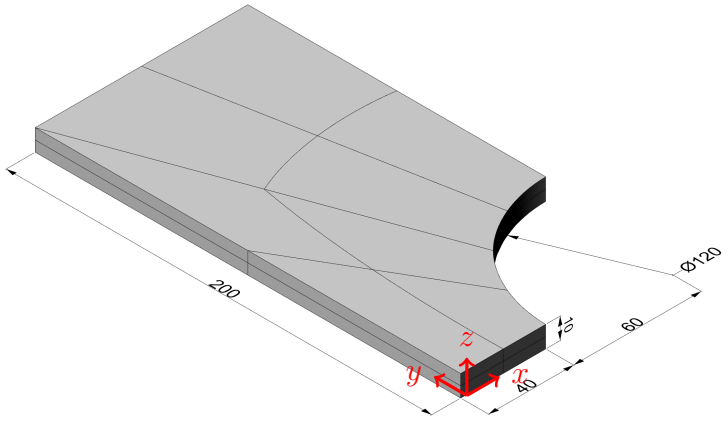
This example consists of a perforated plate subject to uniaxial tension (see Figure 5.8a). The geometry is defined by the open-knot vectors and control points given in Appendix B. Symmetry boundary conditions are applied on the planes at $x = 100mm$, $y = 0mm$, $z = 0mm$. The boundary at $y = 200mm$ is uniformly displaced by $9.5mm$ in the y-direction. The displacement increments are shown in Figure 5.9c.

The problem is solved with the material parameters listed in Table 5.8 on a static mesh obtained by degree elevation on the base geometry to obtain a uniform degree $p = 2$. Successively, 10, 20, and 1 equispaced knots are inserted in x, y, and z-direction, respectively. The base mesh is pre-refined with two refinement levels around the plastification area and four refinement levels around the necking area, as shown in Figures 5.8b and 5.8c. The mesh is of admissibility class one, namely functions belonging to at most two consecutive hierarchical levels can be active on each element Bracco et al. [2019]; Buffa and Giannelli [2016]; Carraturo et al. [2019]. Figures 5.9a and 5.9b show the von Mises stress and equivalent plastic strain $\bar{\alpha}$ after the last displacement step. Figure 5.9c shows the load-displacement curve.

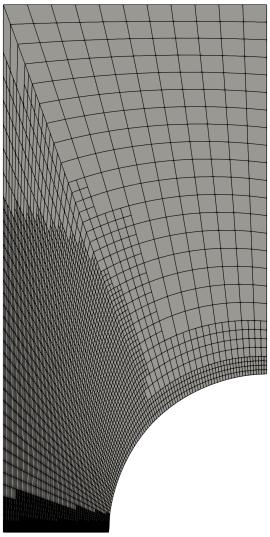
As in the previous section, the CPU time of a C++ implementation of Algorithm 5.5 is compared to a direct extraction $\boldsymbol{C}^e\boldsymbol{B}^e$ for an explicitly constructed extraction operator $\boldsymbol{C}^e$. The measured time includes the matrix multiplication and tensor-product operations listed in Table 5.3, including the Bernstein polynomials' evaluation. The measurements are taken on an `Intel(R) Xeon(R) CPU E5` CPU with a fixed frequency of 2.90 GHz for the first 6 displacement steps. Note that the time for constructing the extraction operator $\boldsymbol{C}^e$ is ignored. The amount of time saved by Algorithm 5.5 is measured as in Equations (5.29) and (5.30), yielding the results in Table 5.9.

p	time_saved	reduction_factor
2	63.9%	2.8
3	90.1%	10.1

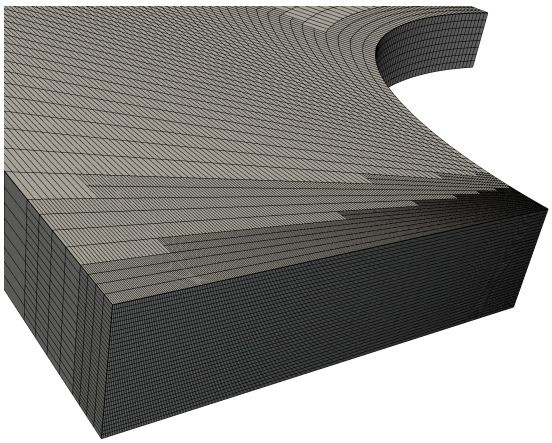
Table 5.9: Finite-strain J2 elastoplasticity example: time saved in extracting the functions through Algorithm 5.5 instead of direct extraction $\boldsymbol{C}^e\boldsymbol{B}^e$ for an explicitly constructed extraction operator $\boldsymbol{C}^e$.



(a) Geometric setup [mm].

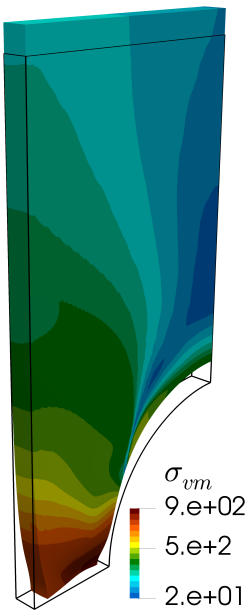


(b) Mesh.

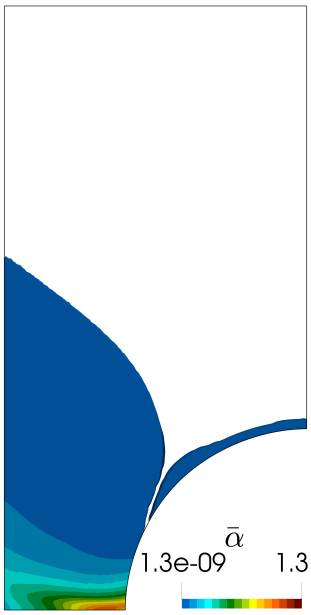


(c) Mesh around necking area.

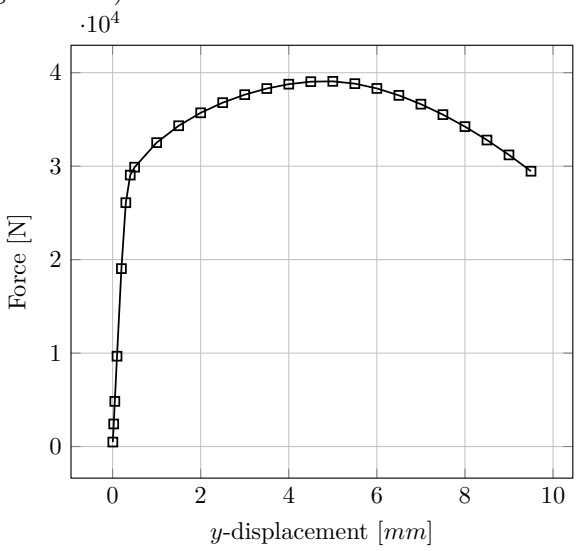
Figure 5.8: Geometry and mesh of the elastoplastic perforated plate.



(a) Final von Mises stress on deformed geometry (warping factor 1).



(b) Final plastified area ($\bar{\alpha} > 0$).



(c) Load-displacement curve.

Figure 5.9: Von Mises, equivalent plastic strain and load-displacement curve obtained on the elastoplastic perforated plate.

Chapter 6

Local refinement for the finite cell method and trimming

The previous chapters introduced a local refinement formulation suitable for B-splines and NURBS shape functions. In this chapter, the hierarchical NURBS are combined with the finite cell method (FCM) [Düster et al., 2008, 2017; Parvizian et al., 2007]: a traditionally high-order finite-element approach that is based on meshes that are not fitted to the domain boundary.

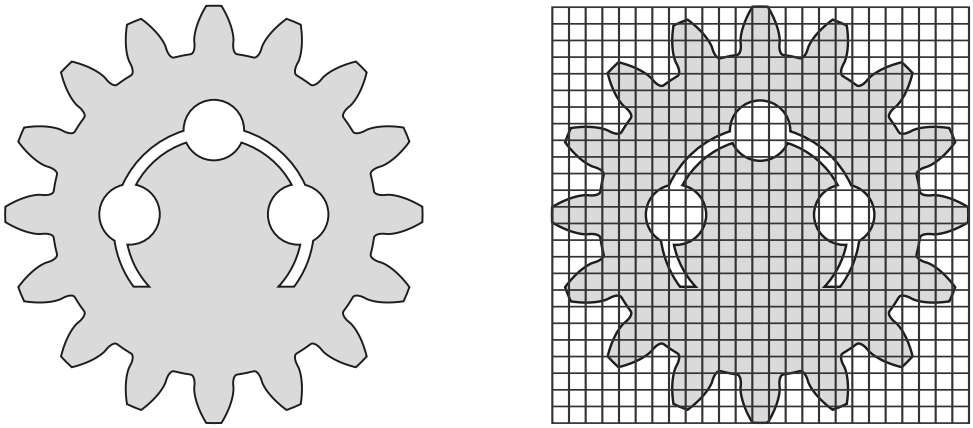
In standard finite elements, local refinement is classically used to efficiently resolve small-scale solution characteristics, obtaining an accuracy similar to a uniform refinement, but using fewer degrees of freedom. This chapter shows that, additionally, it can mitigate FCM-specific difficulties such as

- the unphysical coupling between the sides of a thin hole and
- the overconstraining induced by weak boundary conditions.

6.1 Trimming through the finite cell method

Finite cell analyses are based on meshes that are not conforming to the physical domain Ω. Namely, Ω is contained (or "immersed") in a larger domain Ω^{fict} that can be trivially discretized. For example, Ω^{fict} can consist of the smallest axis-aligned bounding box containing Ω. In this case, Ω^{fict} can be meshed by a regular Cartesian grid of elements, resulting in element faces not matching the physical boundary $\partial\Omega$ (see Figure 6.1). Instead, $\partial\Omega$ can "cut" the elements in an arbitrary manner, requiring additional effort to capture the characteristics of the original geometry Ω and retain the physics of the actual problem. However, this approach avoids the time-consuming and error-prone mesh generation process, shortening the design-though-analysis loop and enabling the analysis of complex topologies demanded by modern design and manufacturing technologies.

The finite cell method can be used in combination with various shape functions, such as Lagrange or integrated Legendre polynomials (see, e.g., Düster et al. [2008, 2017]; Parvizian et al. [2007]), and NURBS (see, e.g., [Coradello et al., 2020b; Hoang et al., 2017; Ruess et al., 2013; Schillinger et al., 2012c, 2016, 2012d; Verhoosel et al., 2015]). The finite



(a) Domain Ω.

(b) Cartesian mesh of the smallest axis-aligned bounding box Ω^{fict} containing Ω.

Figure 6.1: Fundamental concept of the finite cell method.

cell method enables the direct numerical analysis of various geometrical models, such as point cloud domains (see, e.g., Kudela et al. [2020]), constructive solid geometry (CSG) (see, e.g., Wassermann et al. [2017]), flawed volumetric computer-aided designs (CAD) (see, e.g., Wassermann et al. [2019]), and shell models based on boundary representations (B-Rep) (see, e.g., Breitenberger et al. [2015]; Coradello [2016]; Coradello et al. [2020b]; Guo et al. [2018]), and computed tomography scans (CT) (see, e.g., Elhaddad et al. [2018]; Korshunova et al. [2021a,b, 2020]; Yang et al. [2012a,b]). This approach is also suited to parallelism and distributed computations (see, e.g., Jomo et al. [2019, 2017]).

The finite cell method has been applied successfully to various problems of engineering relevance, such as thermoelasticity (see, e.g., Zander et al. [2012]), thermo-elasto-plasticity (see, e.g., Özcan et al. [2019]), additive manufacturing processes (see, e.g., Carraturo et al. [2020]), geometrical nonlinearities (see, e.g., Schillinger [2012]; Schillinger et al. [2012a,b]), contact mechanics (see, e.g., Bog et al. [2015, 2018]), explicit and implicit elastodynamics (see, e.g., Elhaddad et al. [2015]; Joulaian et al. [2014]), bio-mechanics (see, e.g., Elhaddad et al. [2018]; Ruess et al. [2012]; Verhoosel et al. [2015]; Yang et al. [2012a,b]), elastoplasticity (see, e.g., Abedian et al. [2013b, 2014]; Taghipour et al. [2018]), topology optimization (see, e.g., Cai et al. [2014]; Parvizian et al. [2011]), finite-strain problems (see, e.g., Garhuom et al. [2020]), brittle fracture (see, e.g., Hug et al. [2020]; Nagaraja et al. [2019]), quality assurance of additively manufactured products (see, e.g., Korshunova et al. [2021a,b, 2020]), cohesive fracture (see, e.g., Zander et al. [2017]), and foamed materials (see, e.g., Heinze et al. [2018, 2015]).

In its basic form, the finite cell method is based on a weak problem defined on the domain Ω

$$\begin{aligned} &\text{find } u \in \mathcal{S}(\Omega) \\ &\text{such that } a(w, u)_\Omega = l(w)_\Omega \qquad \forall w \in \mathcal{W}(\Omega), \end{aligned} \tag{F}$$

where $\mathcal{S}(\Omega)$ and $\mathcal{W}(\Omega)$ are the trial and test spaces, respectively, while $a(w,\, u)_\Omega$ and $l(w)_\Omega$ represent the bilinear and linear forms modeling the physics of the problem. The subscript refers to the integration domain Ω. For example, for Poisson's problem, these forms can be defined as

$$a(w,\, u)_\Omega = (\nabla w,\, \nabla u)_\Omega = \int_\Omega \nabla w \cdot \nabla u \, \mathrm{d}\Omega, \tag{6.1}$$

$$l(w)_\Omega = (w,\, f)_\Omega = \int_\Omega w\, f \, \mathrm{d}\Omega, \tag{6.2}$$

for some function $f : \Omega \to \mathbb{R}$. This weak form is further discussed in Chapter 7.

The finite-dimensional subspaces for the finite cell method can be defined using a fictitious domain Ω^{fict} containing the physical domain $\Omega \subset \Omega^{\text{fict}}$. The domain Ω^{fict} can be chosen of a shape that can be trivially meshed (see Figure 6.1). A suitable finite-dimensional space $\mathcal{W}^h(\Omega^{\text{fict}})$ is defined on such a mesh. For example, $\mathcal{W}^h(\Omega^{\text{fict}})$ can be spanned by a finite number of NURBS or piecewise polynomials defined on a parameter space $\hat{\Omega}^{\text{fict}}$ combined with the geometrical mapping $\Omega^{\text{fict}} = \boldsymbol{F}(\hat{\Omega}^{\text{fict}})$ (see Section 2.5). It is assumed that the functions in $\mathcal{W}^h(\Omega^{\text{fict}})$ have non-empty support on Ω, namely

$$supp\left(w^h\right) \cap \Omega \neq \emptyset, \qquad \forall w^h \in \mathcal{W}^h(\Omega^{\text{fict}}). \tag{6.3}$$

A discrete subspace for Problem (F) can be defined as

$$\mathcal{W}^h(\Omega) = span\left\{ w^h\big|_\Omega \,:\, w^h \in \mathcal{W}^h(\Omega^{\text{fict}}) \right\}, \tag{6.4}$$

determining the following Galerkin finite cell problem

$$\begin{aligned} &\text{find } u^h \in \mathcal{W}^h(\Omega), \\ &\text{such that } a(w^h,\, u^h)_\Omega = l(w^h)_\Omega, \qquad \forall w^h \in \mathcal{W}^h(\Omega). \end{aligned} \tag{6.5}$$

Note that the bilinear and linear forms are still defined on Ω (see, e.g., Equations (6.1) and (6.2)). Specifically, the integrals are computed on the physical domain Ω and not on Ω^{fict}. However, since the basis functions are defined on Ω^{fict}, it can be convenient to express the integrals over Ω in terms of integrals over Ω^{fict} through a domain-indicator function $\alpha : \Omega^{\text{fict}} \to [0, 1]$

$$\alpha(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega, \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

For example, Equation (6.1) can be written as

$$a(w^h,\, u^h)_\Omega = (\nabla w^h,\, \nabla u^h)_\Omega, \tag{6.7}$$

$$= (\nabla w^h,\, \alpha \nabla u^h)_{\Omega^{\text{fict}}}. \tag{6.8}$$

The domain-indicator function α penalizes the value of the integrand outside the physical domain, recovering the physics of the problem. However, a discontinuity is introduced in

the integrands, requiring non-standard integration rules to retain accuracy (cf., e.g., Abedian et al. [2013a]; Breitenberger et al. [2015]; Hubrich et al. [2017]; Joulaian et al. [2016]; Kudela et al. [2015, 2016]; Marussig and Hughes [2018]; Müller et al. [2013]; Parvizian et al. [2007]; Rank et al. [2012]). See Marussig and Hughes [2018] for a comprehensive review.

This immersed approach can result in very ill-conditioned system matrices when some basis functions have a tiny portion of their support belonging to the physical domain (see, e.g., de Prenter et al. [2017]). One possible way to stabilize the problem is to modify the definition of α to

$$\alpha(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega, \\ \epsilon & \text{otherwise,} \end{cases} \tag{6.9}$$

for a small value $0 < \epsilon \ll 1$. This modification corresponds to solving a perturbed weak problem

$$\begin{aligned} &\text{find } u \in \mathcal{S}(\Omega) \\ &\text{such that } a(w,\, u)_{\Omega} + \epsilon\, a(w,\, u)_{\Omega^{\text{fict}} \setminus \Omega} = l(w) \qquad \forall w \in \mathcal{W}(\Omega). \end{aligned} \tag{6.10}$$

This approach introduces a modeling error of order $O(\sqrt{\epsilon})$ in the energy norm (see Dauge et al. [2015]; Stolfo et al. [2019]).

6.2 The finite cell method for trimmed Kirchhoff-Love shells

The weak formulation of the Kirchhoff-Love shell problem with weak boundary conditions reads as follows (see., e.g., Apostolatos et al. [2015]; Cirak [2006]; Coradello et al. [2020b]; Guo and Ruess [2015]; Herrema et al. [2019]; Kiendl et al. [2009])

$$\begin{aligned} &\text{find } \boldsymbol{u} \in \mathcal{H}^2(\Omega), \\ &\text{such that } a(\boldsymbol{w},\, \boldsymbol{u}) + b^{\text{disp}}(\boldsymbol{w}, \boldsymbol{u}) + b^{\text{rot}}(\boldsymbol{w}, \boldsymbol{u}) = l(\boldsymbol{w}), \qquad \forall \boldsymbol{w} \in \mathcal{H}^2(\Omega), \end{aligned} \tag{6.11}$$

where $a(\boldsymbol{w}, \boldsymbol{u})$ is the bilinear form representing the internal work

$$a(\boldsymbol{w}, \boldsymbol{u}) = \int_{\Omega} \boldsymbol{\varepsilon}(\boldsymbol{w}) \,:\, \boldsymbol{N}(\boldsymbol{u})\, \mathrm{d}\Omega + \int_{\Omega} \boldsymbol{\kappa}(\boldsymbol{w}) \,:\, \boldsymbol{M}(\boldsymbol{u})\, \mathrm{d}\Omega.$$

The symbols $\boldsymbol{\varepsilon}$ and $\boldsymbol{\kappa}$ denote the membrane and bending strain tensors, respectively, while $\boldsymbol{N}$ and $\boldsymbol{M}$ are the in-plane stress and bending moment. Following the FCM approach explained in Section 6.1, this bilinear form can be defined on a trivially-meshable embedding domain Ω^{fict}

$$a(\boldsymbol{w}, \boldsymbol{u}) = \int_{\Omega^{\text{fict}}} \boldsymbol{\varepsilon}(\boldsymbol{w}) \,:\, \alpha \boldsymbol{N}(\boldsymbol{u})\, \mathrm{d}\Omega + \int_{\Omega^{\text{fict}}} \boldsymbol{\kappa}(\boldsymbol{w}) \,:\, \alpha \boldsymbol{M}(\boldsymbol{u})\, \mathrm{d}\Omega,$$

where α is defined in Equation (6.6). A stabilized problem can be obtained by approximating α by Equation (6.9).

The term $l(\boldsymbol{w})$ is the linear functional representing the external work of a volumetric body load $\boldsymbol{f}$ and the traction $\boldsymbol{t}$ over the boundary $\Gamma_t \subset \partial\Omega$

$$l(\boldsymbol{w}) = \int_{\Omega} \boldsymbol{w} \cdot \boldsymbol{f} \, \mathrm{d}\Omega + \int_{\Gamma_t} \boldsymbol{w} \cdot \boldsymbol{t} \, \mathrm{d}\Gamma.$$

For simplicity, only zero external bending moments are considered. The term $b^{\mathrm{disp}}(\boldsymbol{w}, \boldsymbol{u})$ penalizes a displacement different than $\boldsymbol{g}$ on the boundary $\Gamma_g \subset \partial\Omega$

$$b^{\mathrm{disp}}(\boldsymbol{w}, \boldsymbol{u}) = \int_{\Gamma_g} \beta^{\mathrm{disp}} \, \boldsymbol{w} \cdot (\boldsymbol{u} - \boldsymbol{g}) \, \mathrm{d}\Gamma,$$

where $\beta^{\mathrm{disp}} \in \mathbb{R}$ is a user-defined penalty parameter. Finally, the term $b^{\mathrm{rot}}(\boldsymbol{w}, \boldsymbol{u})$ penalizes the normal rotations on the boundary $\Gamma_\theta \subset \partial\Omega$

$$b^{\mathrm{rot}}(\boldsymbol{w}, \boldsymbol{u}) = \int_{\Gamma_\theta} \beta^{\mathrm{rot}} \, \left(\boldsymbol{n} \cdot \boldsymbol{\Phi}(\boldsymbol{w})\right) \left(\boldsymbol{\Phi}(\boldsymbol{u}) \cdot \boldsymbol{n}\right) \mathrm{d}\Gamma,$$

where $\beta^{\mathrm{rot}} \in \mathbb{R}$ is a user-defined penalty parameter, the symbol $\boldsymbol{n}$ represents the outward in-plane normal to the boundary Γ_θ, and $\boldsymbol{\Phi}(\boldsymbol{u}) = \boldsymbol{a}_3(\boldsymbol{u}) - \boldsymbol{A}_3$ denotes the angle between the shell normal in its undeformed ($\boldsymbol{A}_3$) and deformed ($\boldsymbol{a}_3(\boldsymbol{u})$) configuration after applying the deformation $\boldsymbol{u}$. See Guo and Ruess [2015]; Herrema et al. [2019] for a detailed review.

Following Herrema et al. [2019], given the Young's modulus E, Poisson's ratio ν, thickness t, and the smallest-element size h, the penalty parameters are scaled as

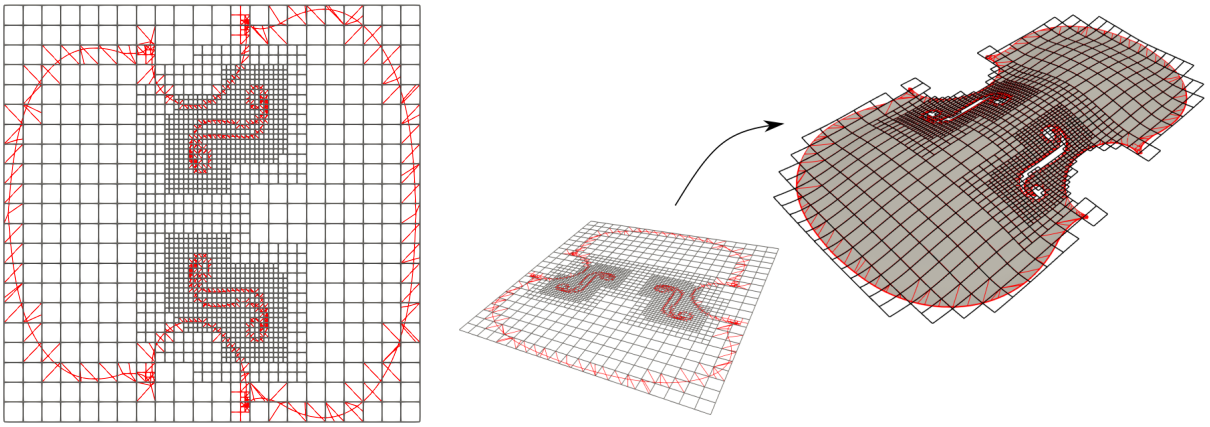
$$\beta^{\mathrm{disp}} = \bar{\beta} \frac{Et}{h(1-\nu^2)} \tag{6.12}$$

$$\beta^{\mathrm{rot}} = \bar{\beta} \frac{Et^3}{12h(1-\nu^2)}, \tag{6.13}$$

where the common parameter $\bar{\beta} \in \mathbb{R}$ is user-defined.

6.2.1 Numerical integration

The shell geometry Ω is assumed to be described by a NURBS patch, as in standard CAD models based on B-Reps. In particular, the geometry is defined by a mapping $\boldsymbol{F}$ relating the patch parameter space $\hat{\Omega}$ to the shell domain $\Omega = \boldsymbol{F}(\hat{\Omega})$. For shells represented by surfaces in three-dimensional space, the parameter space is assumed to be two-dimensional, as shown in Figure 6.2. Accurate numerical integration of the system matrices is obtained following Rank et al. [2011], where quadrature rules are constructed in the parameter space $\hat{\Omega}$. Specifically, the trimming curves are defined in $\hat{\Omega}$, allowing to accurately integrate trimmed elements through the definition of boundary-conforming



(a) Patch parameter space.

(b) Mapping from parametric to physical space. Only the elements intersecting the physical domain are shown.

Figure 6.2: Illustration of the locally-refined elements (black lines) and integration domains (red lines) in the patch parameter space and their mapping to physical space.

integration subdomains, as described in Kudela et al. [2015, 2016]. The method can treat complex trimming curves including sharp features and handles directly B-Rep models produced by standard CAD software such as *Rhinoceros*[5], in the spirit of a design-through-analysis workflow. This approach is applied to isogeometric trimmed shells analysis in, e.g., Breitenberger et al. [2015]; Coradello [2016]; Coradello et al. [2020b]; Guo et al. [2018].

6.3 Motivation to local refinement[c]

The use of hierarchical NURBS for the analysis of trimmed geometries is motivated by the following arguments:

- removal of unphysical coupling between the sides of a thin hole;
- increased accuracy close to trimming boundaries subject to weak constraints;
- efficient approximation of localized deformations.

6.3.1 Thin holes

When the trimming curves define holes that are "thin" compared to the geometry knot spans, the support of a basis function intersected with the physical domain Ω may be disconnected and composed of several disjoint sub-domains. For example, Figure 6.4b shows the support of a function with two disjoint physical sub-domains. This configuration creates an artificial coupling between the two sides of the hole, generally resulting in an unphysical mechanical response. These spurious effects become particularly severe when the local behavior of the structure is strongly determined by the geometry of the hole, as is often the case for complex models. A similar observation can be found in Zander [2017].

[5] www.rhino3d.com

Algorithm 6.1: Refinement towards trimming curves

Input: $l_{\max}$: maximum refinement level
Γ: trimming curve
γ: refinement width parameter

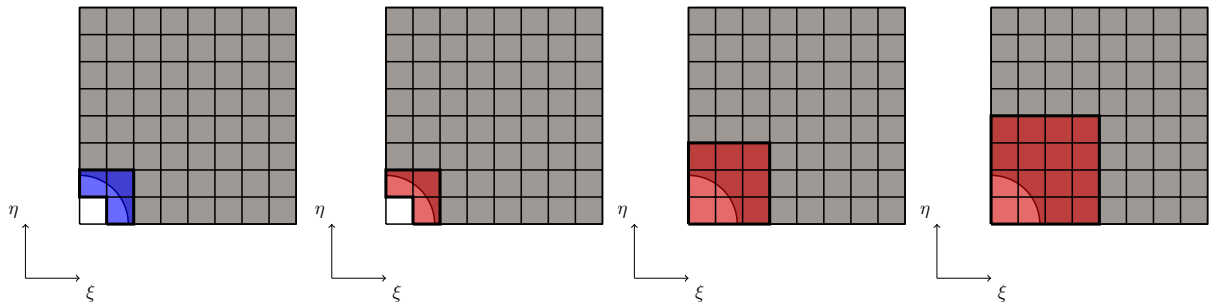
```
for l = 1 ... l_max do
    for each level-l active element e_î^(l) and multi-index î cut by Γ do
        for each level-l active element e_i^(l) and multi-index i do
            if ||i − î||_∞ ≤ γ then
                mark e_î^(l) for refinement
            end
        end
    end
    refine marked elements
end
```

For instance, consider the setup illustrated in Figure 6.4a, where the geometry is modeled via a trimmed NURBS surface of degree $p = 2$, exported from the *Rhinoceros* software together with a set of 32 trimming curves. This geometry features two internal thin holes described by trimming curves presenting sharp features. For the analysis, clamped boundary conditions are applied to the outer boundary of the violin via the penalty method, as described in Section 6.2, and a line load $\boldsymbol{F} = (0, 0, 100)^\top$ is applied to the reentrant tip of the f-hole. When adopting as computational mesh the same NURBS patch defining the geometry (as exported from *Rhinoceros*), this numerical experiment yields an unphysical response. The maximum displacement is attained towards the center of the geometry (see Figure 6.4c obtained with $\bar{\beta} = 10$) instead of the reentrant tip, as expected from engineering intuition.

To mitigate this issue, locally-refined hierarchical NURBS can substitute functions having large support with functions having smaller support. In particular, the unphysical coupling is necessarily removed when no support is composed of disconnected physical subdomains. To this end, let us consider the refinement procedure presented in Algorithm 6.1. The marking parameter γ represents the extension of the refinement area in the proximity of a given trimming curve Γ. Figure 6.3 shows the effects of the different choices $\gamma = 0, 1, 2$.

When using hierarchical NURBS, setting $\gamma \geq p$ assures that only finest-level functions have support on the trimming curve. By comparing the size of the finest-level knot spans and the size of the hole, one can obtain the value for $l_{\max}$ that removes all functions with disconnected supports.

For the current example, Algorithm 6.1 can refine the computational mesh towards the internal f-hole. The decoupling is obtained for $\gamma = p = 2$ and $l_{\max} = 5$, yielding the qualitatively correct response shown in Figure 6.4d, where the maximum deflection occurs at the reentrant tip. Figure 6.5a shows the z-displacement at point A located at one of the



(a) Active cut cells. (b) $\gamma = 0$. (c) $\gamma = 1$. (d) $\gamma = 2$.

Figure 6.3: Influence of marking parameter γ on the refinement. The cut cells are highlighted in blue, whereas the elements marked for refinement after performing Algorithm 6.1 with $\gamma = 0, 1, 2$, respectively, are colored in red.

geometrical kinks of the trimmed boundary (see Figure 6.4b). A sudden improvement in accuracy is obtained for $l_{\max} = 5$, while smaller values yield inaccurate results induced by the unphysical coupling. Figure 6.5a also compares the described locally-refined solution to the solution obtained by a tensor-product refinement constructed by Algorithm 6.1 with $\gamma = 0$ (local tensor-product refinement), where the marked elements are refined using the classic knot-insertion procedure (see Section 2.5.1.1) instead of local refinement. The meshes corresponding to local refinement and local tensor-product refinements are depicted in Figures 6.6a and 6.6b, respectively.

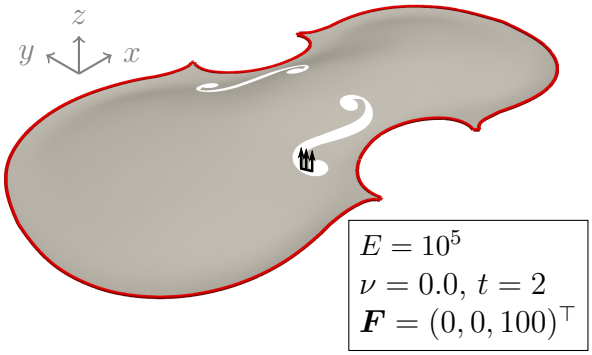
Additionally, Figure 6.5a shows the convergence of a standard uniform tensor-product refinement obtained by recursively bisecting every knot span (uniform tensor-product refinement). Figures 6.5a and 6.5b show that all strategies converge towards a reference value for the displacement obtained from an overkill solution. Local refinement achieves a level of accuracy comparable to uniform refinement using substantially fewer degrees of freedom.

It is observed in Coradello et al. [2020b] that an adaptive procedure driven by the error estimator introduced in Coradello et al. [2020a] automatically removes the unphysical coupling for a simple example. Further research is needed to investigate the robustness of this approach.

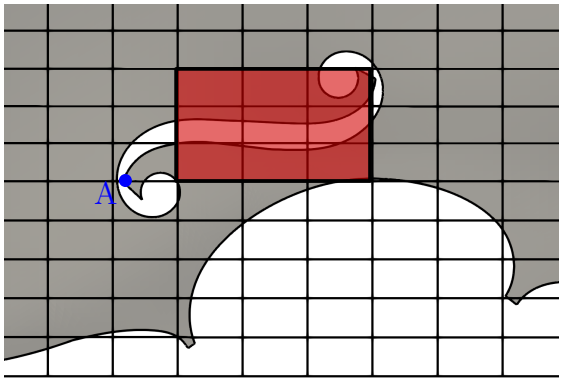
6.3.2 Weak constraints

When imposing weak boundary conditions on a trimming curve, it can happen that the finite element space is not capable of both accurately satisfy the boundary conditions and approximate the numerical solution in the proximity of the trimming boundary. For complicated geometries or high penalty parameters β^{disp} and β^{rot}, some elements will be over-constrained.

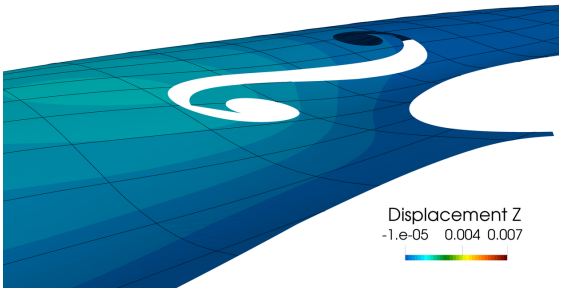
To illustrate this effect, consider the setup depicted in Figure 6.7a, where a uniformly-distributed vertical load $\boldsymbol{F} = (0, 0, -100)^\top$ is applied on the top surface. Clamped boundary conditions are enforced with a penalty term ($\bar{\beta} = 10^6$) on the internal curves featuring



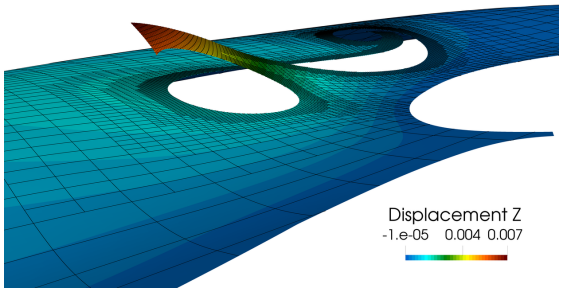
(a) Applied boundary conditions: clamped boundary (red wire) and distributed load (arrows).



(b) Example of basis function support covering both sides of a thin hole ($p = 2$ and maximum continuity).



(c) Unphysical solution obtained from a direct analysis of the CAD model.



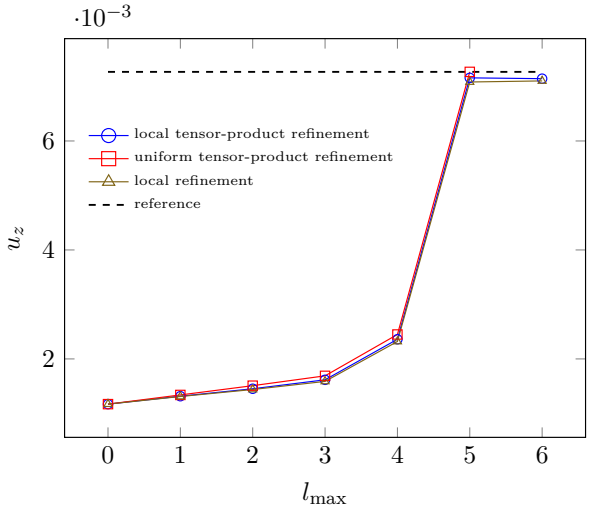
(d) Local refinement around the trimming curve removes the unphysical coupling.

Figure 6.4: Thin-hole example. The discretization of a complex trimmed geometry creates an unphysical coupling between the sides of thin holes.

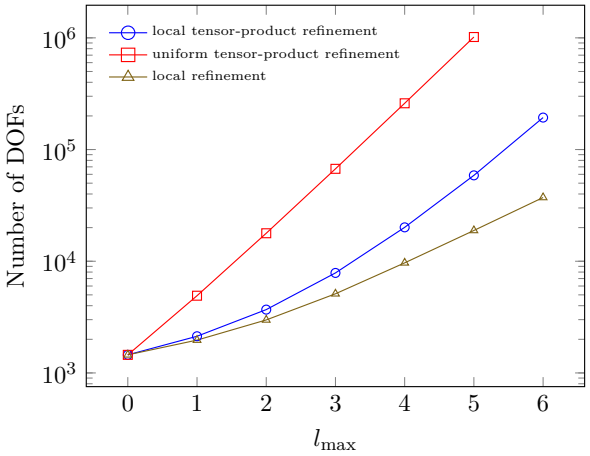
kinks and areas of high curvature. Figures 6.7b and 6.7c show that the displacement and stress are artificially low on the elements cut by the trimming curve, spuriously following the element boundaries.

Given a fixed mesh, an optimal choice for the penalty parameter β can mitigate the described problem. See de Prenter et al. [2018] for a discussion on this matter in the context of Nitsche's method. If it is unknown how to choose β optimally, mesh refinement can improve the accuracy in both interior and boundary terms by enlarging the finite element space and intersecting the curve with smaller elements, generally yielding simpler intersections. This way, small geometric features induced by the trimming curves can be selectively resolved. For instance, Figures 6.7d and 6.7e show the solution and Von Mises stress distribution produced by Algorithm 6.1 with $\gamma = p = 2$ and $l_{\max} = 5$. This example qualitatively demonstrates the capability of local refinement to reduce the over-constraining effects linked to the weak imposition of Dirichlet-type boundary conditions.

Figure 6.8 compares the resultant Von Mises stress for different meshes. Figure 6.8b shows the solution obtained through a uniform mesh of 38 607 DOFs. The mesh is chosen to have a number of DOFs similar to the locally-refined mesh in Figures 6.7e and 6.8a, which

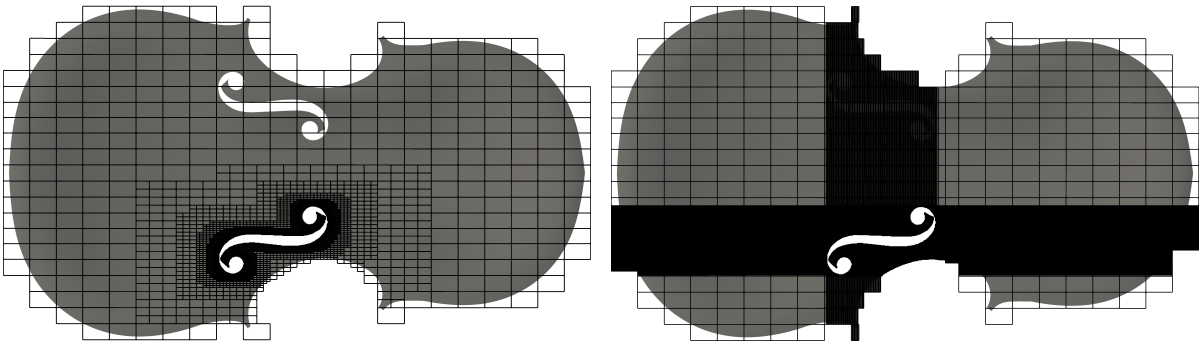


(a) z-displacement at point A. A sudden change is observed for $l_{\max} = 5$.



(b) Number of DOFs.

Figure 6.5: Thin-hole example. Displacement and number of DOF.



(a) Locally-refined mesh.

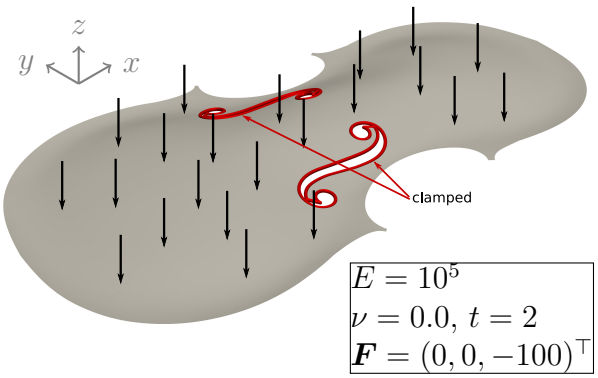
(b) Local tensor-product mesh.

Figure 6.6: Meshes obtained by local refinement and local tensor-product refinement produced by Algorithm 6.1 with $\gamma = p = 2$ and $l_{\max} = 5$.

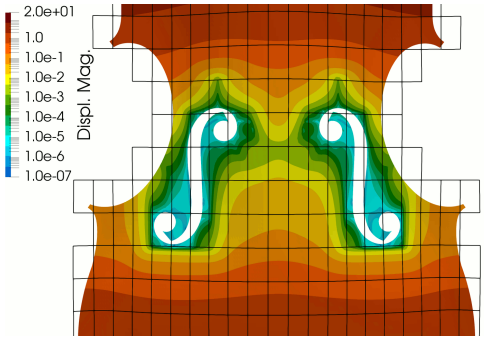
has 36 225 DOFs. As a reference, Figure 6.8c shows the solution obtained by an overkill mesh with 4 026 378 DOFs. The locally-refined mesh does not have artificially low stress on the constrain curve, similarly to the overkill solution.

6.3.3 Localized deformations

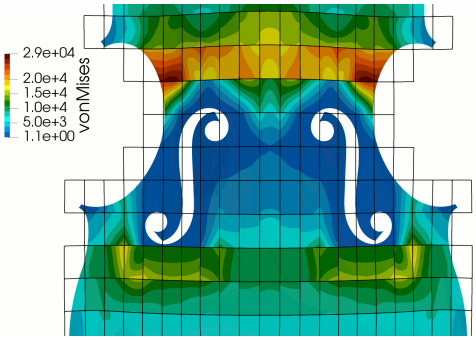
The example given in Figures 6.4a and 6.4d produces a localized deformation that needs to be adequately resolved. The local refinement strategy employed to remove the unphysical coupling in Section 6.3.1 gives an approximation comparable to uniform refinement but with a considerable reduction in DOFs (see Figures 6.5a and 6.5b). Table 6.1 shows the energy and z-displacement errors with respect to the following reference values $\tilde{E} \approx 0.2211$, $\tilde{u}_z \approx 7.268 \cdot 10^{-3}$. These values are extracted from an overkill solution defined on a uniformly-refined mesh obtained by $l_{\max} = 5$ recursive bisection steps.



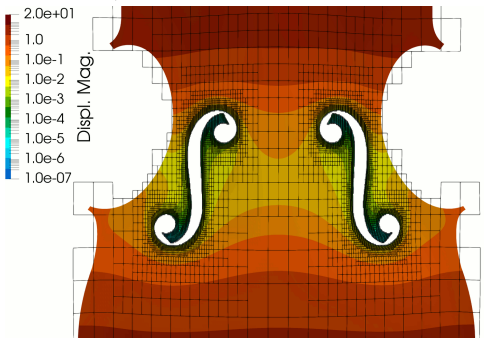
(a) Boundary conditions: clamped (red wire) and distributed load (arrows).



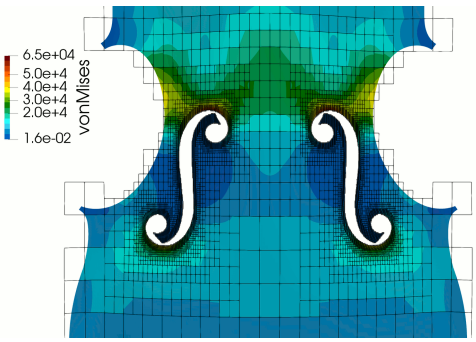
(b) Displacement magnitude (logarithmic scale) on the unrefined mesh.



(c) Von Mises stress on the unrefined mesh.



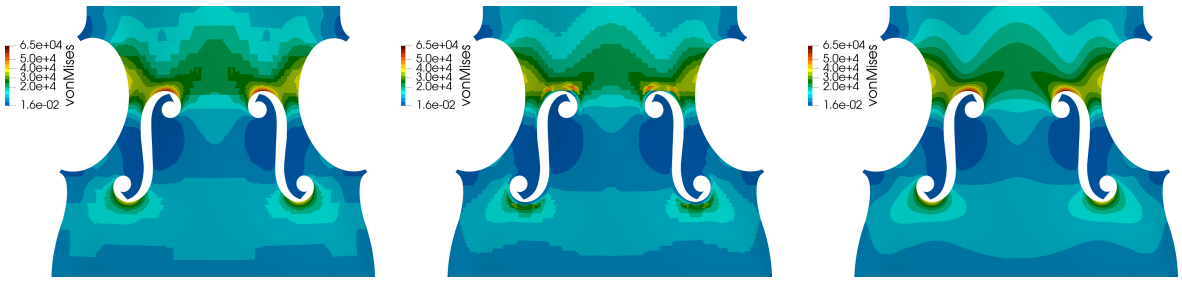
(d) Displacement magnitude (logarithmic scale) on a locally-refined mesh.



(e) Von Mises stress on a locally-refined mesh.

Figure 6.7: Example of overconstraining induced by weak boundary conditions on trimming curves.

This numerical experiment confirms that local refinement can accurately and efficiently capture local quantities (such as the solution at point A) as well as global quantities (the energy of the system) comparable to the tensor product refinement. Note that, in this particular example, ~ 5 times fewer DOFs are required to achieve similar accuracy.



(a) Von Mises on a locally-refined mesh as in Figure 6.7e.

(b) Von Mises on a uniformly-refined mesh such that the number of DOFs is similar to Figure 6.8a.

(c) Von Mises on an overkill mesh.

Figure 6.8: Overconstraining on trimming curves can be resolved by local refinement.

	num. DOFs	energy error $(1 - \frac{E^h}{\tilde{E}})\%$	z-displ. error at A $(1 - \frac{u_z^h}{\tilde{u}_z})\%$
tensor product ref.	193 359	2.07%	1.72%
local ref.	37 215	2.88%	2.29%

Table 6.1: Comparison of the error in the energy norm and z-displacement at point A against the number of DOFs for tensor product and local refinements.

6.4 Numerical Examples[c]

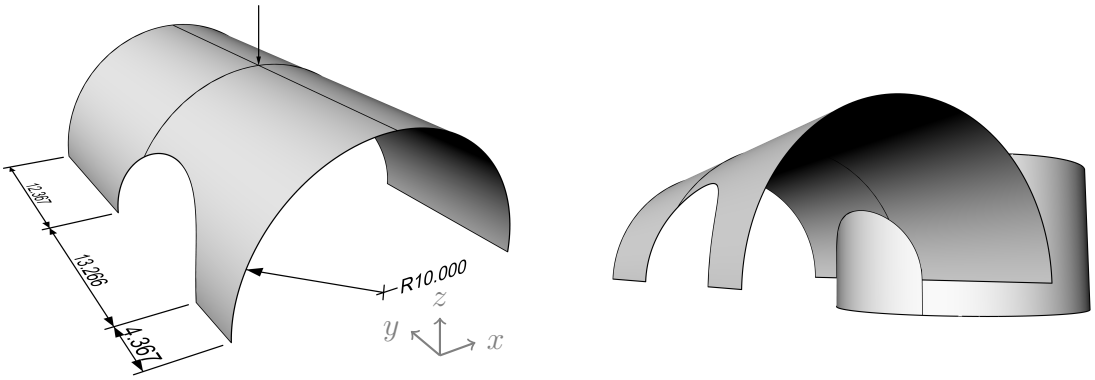
This chapter concludes with two numerical examples that show the applicability of the proposed workflow to the locally-refined analysis of trimmed surfaces. All the geometries used in the following examples have been created in the commercial CAD software *Rhinoceros* and exported in the STEP file format [ISO 10303-11:1994, 1994].

6.4.1 A trimmed adaptive example

The first example deals with a geometry created by the intersection of two cylinders, as shown in Figure 6.9. Such a geometry is taken from an example presented in Casquero et al. [2017]. Note that the two trimmed holes are symmetric to the diagonal of the cylinder. The two circular ends of the cylinder on the xz-plane are subject to no-displacement boundary conditions. A unit point load $\boldsymbol{F} = (0, 0, -1)^\top$ is applied on the surface middle point (see Figure 6.9). The Young modulus and Poisson's ratio parameters are chosen as $E = 5 \cdot 10^7$ and $\nu = 0.0$, respectively. The shell thickness is set equal to $t = 0.1$.

The $\mathcal{L}^\infty$-norm of the displacement gradient is used as a simple indicator for adaptively refining the mesh. In particular, given a user-defined threshold $\gamma \in (0, 1)$, one element Ω^e is marked for refinement if

$$||\nabla \mathbf{u}^h||_{\mathcal{L}^\infty(\Omega^e)} > \gamma \max_{\tilde{\Omega}^e} ||\nabla \boldsymbol{u}^h(\tilde{\Omega}^e)||_{\mathcal{L}^\infty(\tilde{\Omega}^e)},$$



(a) Geometry definition in Rhino. (b) Trimming boolean operation.

Figure 6.9: Geometry specifications and representation of the trimming operation in Rhino used to create the two trimmed holes, defined as the intersection between two cylinders.

where the maximum is taken over all elements $\tilde{\Omega}^e$ of the mesh. Additionally, the elements adjacent to the supports are always marked for refinement. Despite the simplicity of this error indicator, numerical experiments show that it behaves well in driving an adaptive simulation for the problem at hand, capturing the main characteristics of the solution (see Figure 6.10).

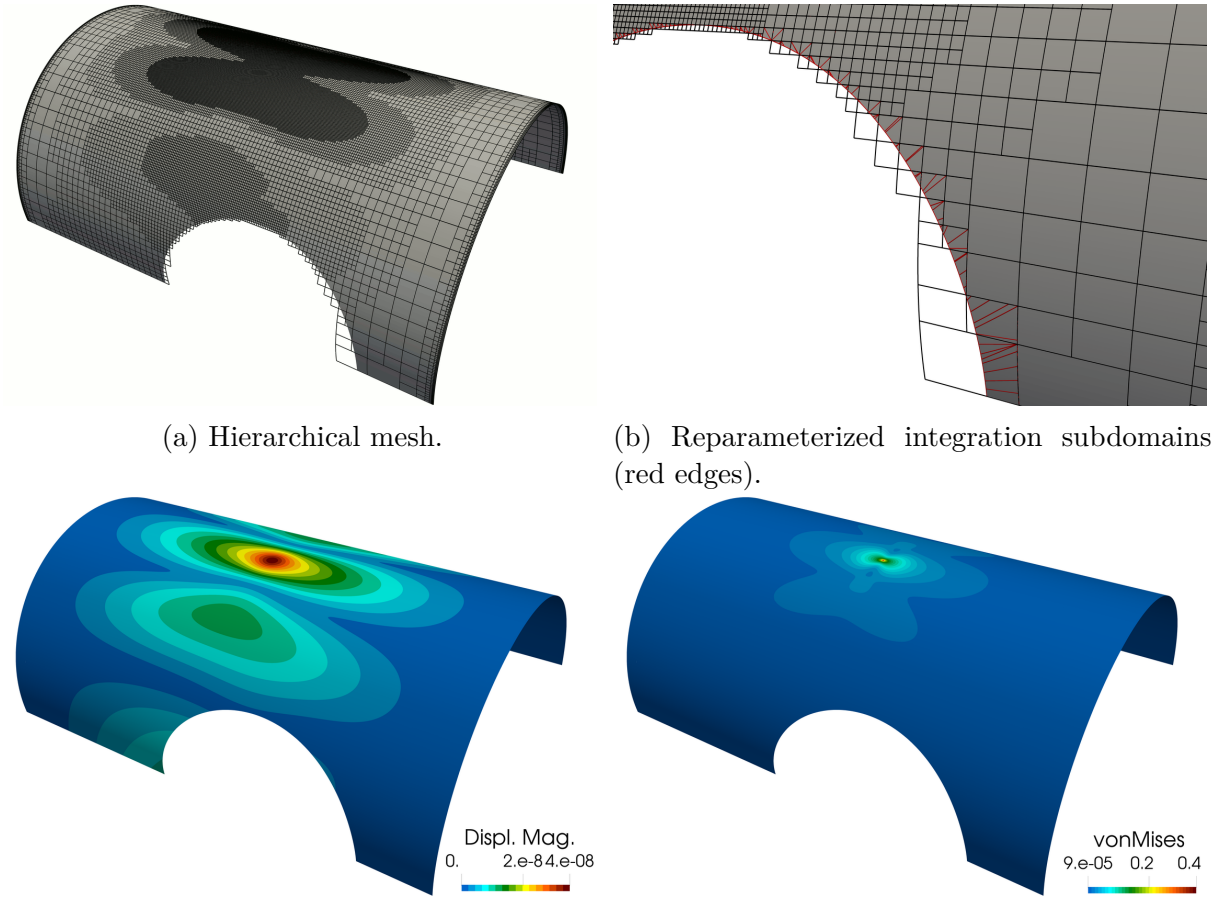
Figure 6.11 shows the convergence of the internal energy and z-displacement under the point load on a series of refined meshes. The reference solution consists of an overkill solution obtained on a uniformly-refined mesh with 2 603 682 DOFs, $p = 3$.

Figure 6.10b shows the reparameterized integration cells in the proximity of one of the trimming curves. Note that the reparameterized integration is used on trimmed elements belonging to different hierarchical levels. Often, the local refinement simplifies the intersections between the trimming curve and the elements, yielding fewer intersection cases to be handled.

6.4.2 B-Rep analysis with weak constraints

The following example concerns the simulation of a simplified model of the Rolex Learning Center rooftop (see Figure 6.12a), the campus library at the École Polytechnique Fédérale de Lausanne (see Figure 6.12b).

The structure is modeled by a trimmed NURBS surface of degree $p = 3$ composed of 20×20 knot spans. A planar map of the building has been used to trace all the major structural holes. The roof is supported by pillars modeled by 150 cylinders arranged in a 15×10 Cartesian fashion into the model (those falling outside of the physical domain have been discarded). Their intersection with the surface is used to impose homogeneous displacement boundary conditions ($\bar{\beta} = 10^3$). This simplified design may not correspond to the actual placement of the pillars. The material parameters are chosen as $E = 40 \cdot 10^9$, $\nu = 0.15$, and $t = 0.2$ for the Young modulus, Poisson ratio, and shell thickness, respectively. Note that the purpose of this example is not to model the static state of the



(a) Hierarchical mesh.

(b) Reparameterized integration subdomains (red edges).

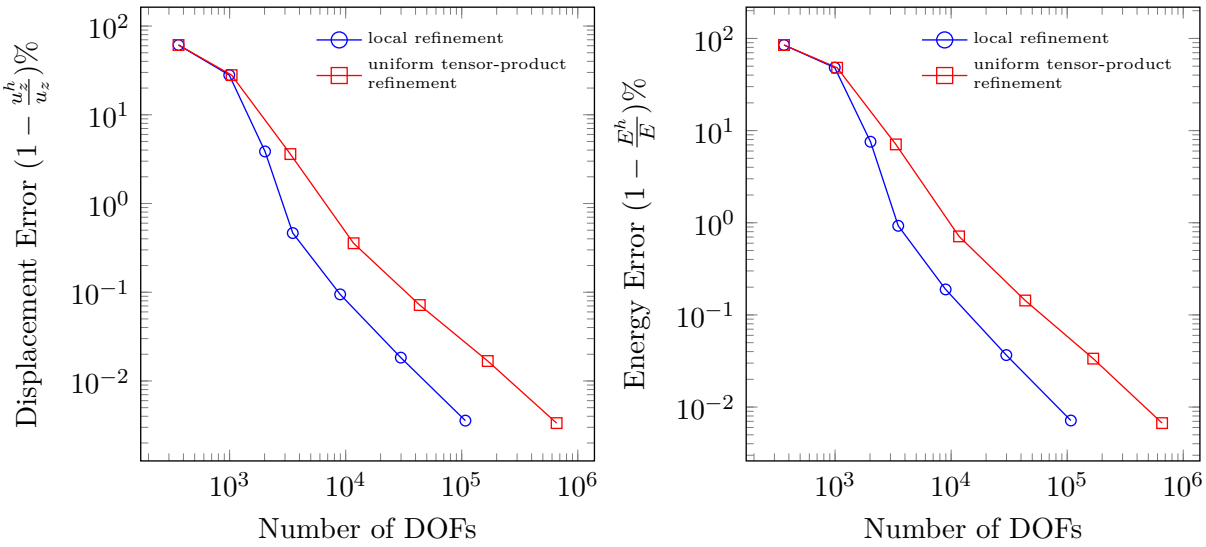
(c) Displacement magnitude.

(d) Von Mises stress.

Figure 6.10: Mesh, displacement magnitude, and Von Mises stress obtained after 6 iterations of the adaptive loop, marking parameter $\gamma = 0.2$. The solution is obtained with cubic hierarchical NURBS.

roof comprehensively but rather to provide an illustrative example of the implemented framework with a complex geometrical model.

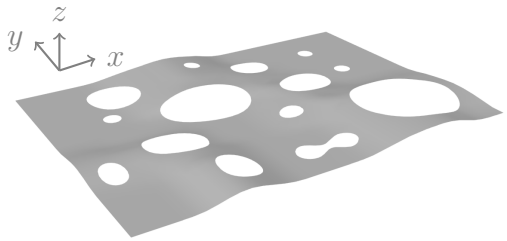
The roof is subjected to a body load $\boldsymbol{F} = (0, 0, -2 \cdot 10^3)^\top$, resulting in the displacement field is depicted in Figure 6.12c, where $k = 5$ hierarchical-refinement levels are used to increase the resolution around the pillars and trimming curves. A close-up of the solution in the vicinity of a trimming curve for a coarse uniform tensor-product mesh and a locally-refined mesh is depicted in Figure 6.13. This example motivates the use of local refinement to selectively resolve the small-scale features of the problem (the pillars). It is also shown how all relevant information needed to impose boundary conditions can be taken directly from the CAD model in the spirit of a complete design-through-analysis workflow.



(a) Error in the z-displacement under the point load.

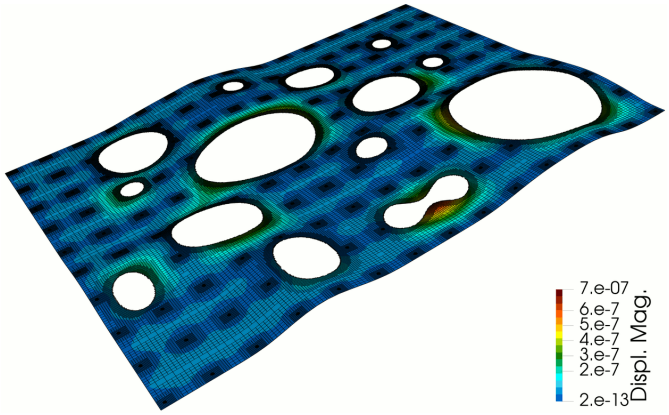
(b) Error measured in the energy norm.

Figure 6.11: Convergence of the error in displacement and energy obtained with cubic hierarchical NURBS and marking parameter $\gamma = 3$.



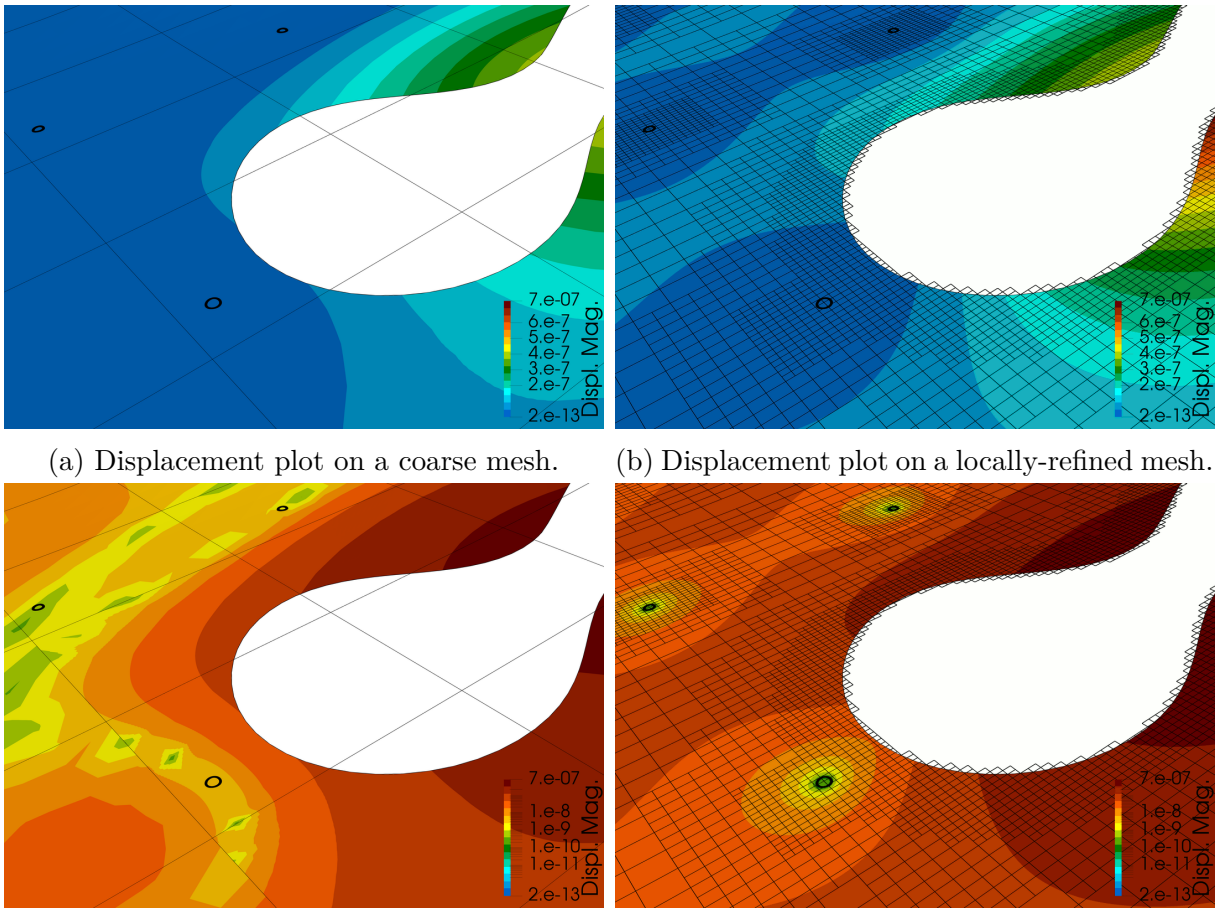
(a) B-Rep model

(b) Aerial photo (source: Wikipedia)



(c) Displacement magnitude.

Figure 6.12: Rolex Learning Center example. The geometric model, the actual building, and the numerical solution of the roof subject to its self-weight. The solution is obtained using cubic hierarchical NURBS with $k = 5$ refinement levels.



(a) Displacement plot on a coarse mesh.

(b) Displacement plot on a locally-refined mesh.

(c) Displacement plot on a coarse mesh (logarithmic scale).

(d) Displacement plot on a locally-refined mesh (logarithmic scale).

Figure 6.13: Rolex Learning Center example. Zoom on the solution in the proximity of a trimming curve. Weak boundary conditions and geometric features are efficiently resolved by local refinement.

Chapter 7

Reactions on trimmed locally-refined meshes

The finite element method is classically based on nodal Lagrange basis functions defined on conforming meshes. In this context, total reaction forces are commonly computed from the so-called "nodal forces", yielding higher accuracy and convergence rates than reactions obtained from the differentiated primal solution ("direct" method).

The finite cell method (FCM) was introduced in the previous chapter as a direct approach to numerical simulation of trimmed geometries. However, body-unfitted meshes preclude the use of classic nodal reaction algorithms.

This chapter shows that the direct method can perform particularly poorly for immersed methods. Instead, conservative reactions can be obtained from equilibrium expressions given by the weak problem formulation, yielding the superior accuracy and convergence rates typical of nodal reactions.

This approach is also extended to basis functions not based on the concept of nodes, such as the (truncated) hierarchical B-splines introduced in Chapters 2 and 3.

7.1 Introduction

In many practical applications, the goal of finite-element analyses is to approximate specific physical quantities of interest. These data are often derived from the primal solution, such as in the case of total reaction forces or fluxes. Such quantities are often the most relevant data in engineering design and analysis.

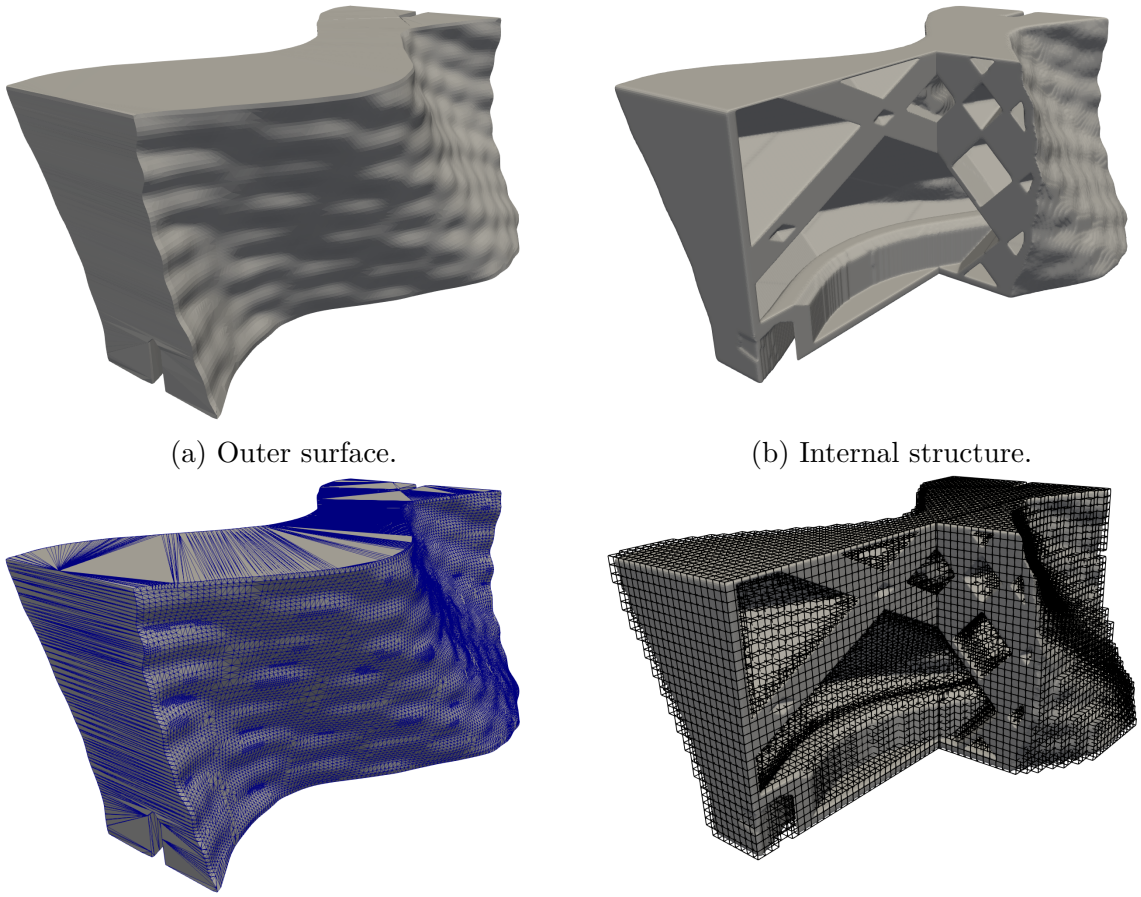
The evaluation of fluxes and forces derived from the primal finite-element solution has been investigated for conforming meshes in several literature contributions. For example, in Akira [1986]; Barrett and Elliott [1987]; Brezzi et al. [2001]; Carey [1982]; Carey et al. [1985]; Gresho et al. [1987]; Hughes et al. [1987]; Hughes [2000]; Hughes et al. [2000]; Oshima et al. [1998] the flux is obtained through a modified variational problem with an additional auxiliary field corresponding to the normal flux over the Dirichlet boundary. Such an approach amounts to a mere post-processing step, and the resulting flux will

fulfill equilibrium in a global or local sense. This technique, referred to as conservative or consistent, is proven in the above references to be more accurate and achieve higher convergence orders than the "direct" approach of differentiating the primal solution. In Melbø and Kvamsdal [2003], reactions on mesh boundaries (subject to strong boundary conditions) are obtained for the Stokes flow through a variational interpretation similar to the one discussed in this work. In van Brummelen et al. [2011], similar formulas for the reactions on (conforming) mesh boundaries are studied, focusing on coupled problems. In the mentioned publications, the reactions are computed on Dirichlet-boundaries of meshes conforming to the computational domain. In Bazilevs and Hughes [2007], this approach is extended to computing reactions on (conforming) mesh boundaries subject to weak boundary conditions. In Bazilevs et al. [2012]; Kamensky et al. [2017]; Kamensky [2016]; Wu et al. [2017], consistent forces on immersed boundaries are considered on the fluid–structure coupling interface based on an augmented Lagrangian formulation.

In this work, the conservative reactions are reviewed for conforming meshes subject to strong boundary conditions. This approach is then extended to non-conforming trimmed meshes, where the boundary of the geometry does not match the element boundaries. In particular, the total reaction flux is computed on boundaries subject to weak boundary conditions, such as the penalty [Babuška, 1973] and the symmetric Nitsche's [Nitsche, 1971] methods. The computation of the total fluxes for conforming meshes is viewed as testing a variational form with specific test functions, serving as "extraction functions" in the framework of Babuška and Miller [1984]. Namely, the reactions are obtained by the expression of equilibrium given by the weak form, yielding a total flux in global equilibrium with the other fluxes and data of the problem. Reactions are observed to converge with rates two times higher than the energy-norm error for Nitsche's method on a trimmed two-dimensional benchmark with a smooth solution. This phenomenon is often referred to as superconvergence [Babuška and Miller, 1984; Hughes et al., 2000; Szabó and Babuška, 2011; Wahlbin, 1995]. The same convergence rates are obtained for the penalty method, provided that the penalization parameter is suitably scaled.

Moreover, it is shown how this approach can be generalized to bases that do not form a partition of unity and are not based on the concept of "node". For example, this approach is valid for hierarchical B-splines.

The structure of the chapter is as follows. Section 7.2 motivates the conservative approach in computing the reactions. A three-dimensional trimmed example with a complex geometry defined by a Standard Triangle Language (STL) file is considered, showing that the direct method can perform particularly poorly for immersed meshes, as the weak boundary conditions indirectly constrain also the gradient of the solution. Section 7.7 explains how the standard way to compute the reactions can be interpreted as testing a variational form with specific test functions. This point of view serves as a basis to compute conservative reactions on trimmed bases not forming a partition of unity in Sections 7.8 and 7.9. In Section 7.10, it is shown that the method is superconvergent and approximates with higher accuracy the total flux in a smooth two-dimensional problem for both penalty and Nitsche's methods. In Section 7.11, the method is shown to give consistent results for both penalty and Nitsche's methods in the considered three-dimensional trimmed exam-



(a) Outer surface.

(b) Internal structure.

(c) STL file describing the geometry.

(d) Non-conforming computational mesh.

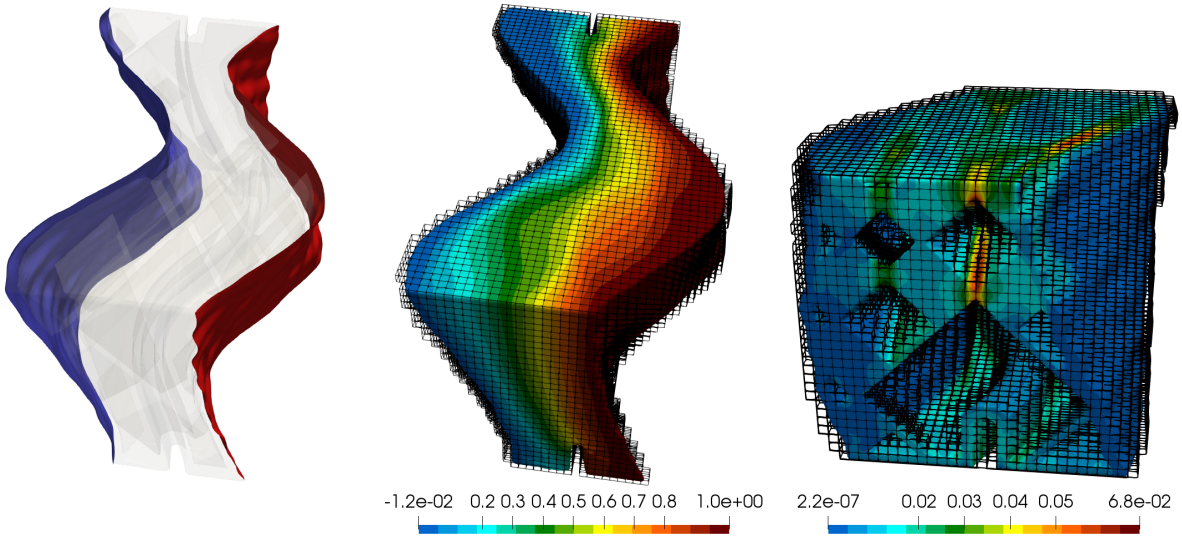
Figure 7.1: Portion of façade element [Mungenast, 2017b].

ple. Finally, Section 7.12 shows how this approach can be applied to compute reaction tractions for isogeometric analysis of trimmed Kirchhoff-Love shells.

7.2 Motivation

Consider the portion of façade element [Mungenast, 2017b] shown in Figure 7.1a. Its design takes advantage of the production freedom offered by additive manufacturing technologies to combine the aesthetics of wavy surfaces with functionalities such as insulation, ventilation, load transfer, and shading (cf. Mungenast [2017a,b]). These functionalities lead to a geometry featuring a complex internal structure and detailed external surfaces (cf. Figure 7.1b). The geometry is described by a fine STL file (courtesy of Dr. Moritz Mungenast), as displayed in Figure 7.1c. Note that the STL file does not define a computational geometry directly suitable for traditional methods based on conforming meshes.

The objective is to compute the total heat flux across the structure induced by a temperature difference on two opposite faces. The following Laplace's equation and boundary



(a) Boundary conditions: $u = 0$ on the blue surface (Γ_0) and $u = 1$ on the red surface (Γ_1).

(b) Temperature.

(c) Heat-flux magnitude on the cross-section.

Figure 7.2: Boundary conditions and solution example for the façade element.

conditions serve as a model problem

$$-\nabla \cdot (\boldsymbol{\kappa} \nabla u) = 0 \qquad \text{in } \Omega, \tag{7.1}$$

$$u = 0 \qquad \text{on } \Gamma_0, \tag{7.2}$$

$$u = 1 \qquad \text{on } \Gamma_1, \tag{7.3}$$

$$\boldsymbol{\kappa} \nabla u \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Omega \setminus \overline{(\Gamma_0 \cup \Gamma_1)}. \tag{7.4}$$

Here, $\Omega \subset \mathbb{R}^3$ denotes the domain defined by the façade element, $\Gamma_0 \subset \partial\Omega$, and $\Gamma_1 \subset \partial\Omega$, $\Gamma_0 \cap \Gamma_1 = \emptyset$, denote the left and right boundaries highlighted in Figure 7.2a, $\boldsymbol{\kappa} \in \mathbb{R}^{3\times3}$ denotes the conductivity tensor, and $\boldsymbol{n}$ the outward unit boundary normal. Following the finite cell approach introduced in the previous chapter, a finite element simulation model is constructed without the need to build a conforming mesh, a potentially time-consuming step in the total simulation pipeline [Cottrell et al., 2009; Hughes et al., 2005]. The geometry Ω is immersed in a larger rectangular cuboid Ω^{fict} that can be straightforwardly meshed by a Cartesian element grid. Figure 7.1d shows an example of elements intersecting the physical domain Ω. Since the boundaries Γ_0 and Γ_1, in general, do not coincide with a subset of element faces, but they are immersed in the elements, a strong imposition of the temperature boundary conditions would significantly deteriorate the accuracy. Instead, these boundary conditions are imposed weakly (cf., e.g., Ruess et al. [2013]; Schillinger et al. [2012c]), as explained in detail in Section 7.5. Figures 7.2b and 7.2c show the temperature and heat flux obtained with B-spline basis functions of order $p = 2$ and $\boldsymbol{\kappa}$ being the identity matrix. The boundary conditions are applied using Nitsche's method with stabilization parameter $\gamma = 10(p+1)^2/h$ (cf. Equation (7.17) and Antolin et al. [2019]; Johansson et al. [2019]), where h denotes the mesh size, as explained in Section 7.5.

A question now arises about the way to accurately compute the total flux from the trimmed finite element solution. Once a numerical solution u^h for the problem of Equations (7.1)–(7.4) is obtained, the conventional way for conforming finite elements with Lagrange shape functions and subject to strong boundary conditions can be summarized as in Table 7.1. It is not clear how the conventional procedure can be used for trimmed

Let $\eta(\Gamma_0)$ be the set of nodes on Γ_0.

1. For each node $A \in \eta(\Gamma_0)$ associated with the nodal shape function N_A, compute the internal nodal flux
$$q_A = \int_\Omega \nabla N_A \cdot \left(\boldsymbol{\kappa} \nabla u^h\right) \, \mathrm{d}\Omega,$$
and the external nodal flux
$$q_A^e = \int_\Omega N_A f \, \mathrm{d}\Omega + \int_{\Gamma_h} N_A h \, \mathrm{d}\Gamma.$$
2. The reaction r on Γ_0 is obtained by summing the nodal fluxes of all nodes located on Γ_0, minus the known external fluxes
$$r = \sum_{A \in \eta(\Gamma_0)} q_A - q_A^e.$$

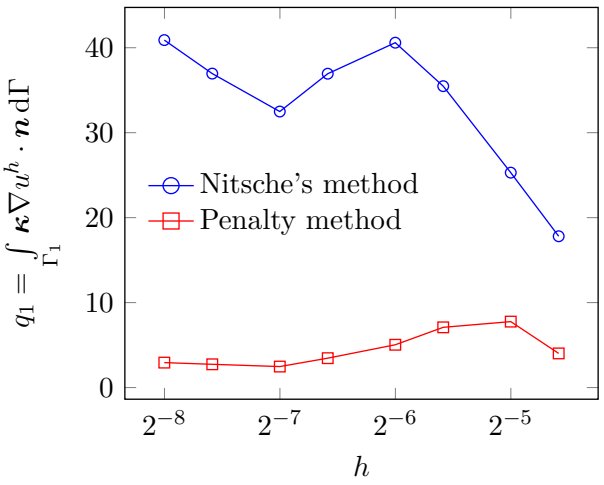
Table 7.1: The traditional algorithm for computing the reactions on conforming meshes of nodal partition-of-unity finite elements [Bathe, 2007; De Borst et al., 2012; Hughes, 2000; Hughes et al., 2000; Kohnke, 2009; Siemens PLM Software Inc, 2014].

meshes with non-nodal shape functions, as there are no nodes and the boundary is immersed in the element domains.

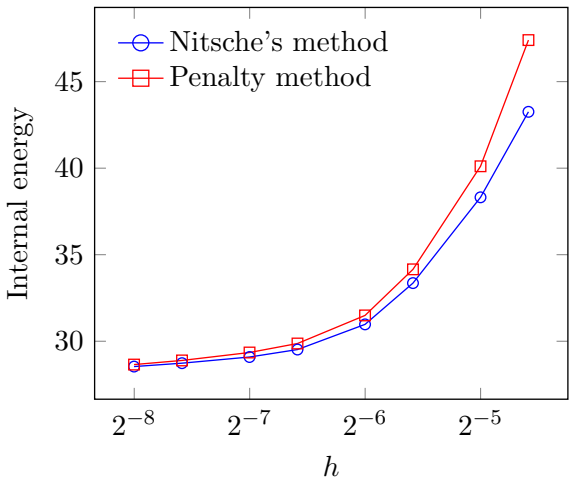
Since the numerical solution u^h defines a numerical flux $\boldsymbol{\kappa}\nabla u^h$ for every spatial location $\boldsymbol{x} \in \Omega$ (u^h is assumed to be at least continuous), it is, in particular, possible to integrate numerically $\boldsymbol{\kappa}\nabla u^h \cdot \boldsymbol{n}$ over Γ_1. However, total fluxes computed in this way can have poor accuracy. Figure 7.3a shows that different total fluxes are obtained for Nitsche's method with stabilization parameter $\gamma = 10(p+1)^2/h$ (cf. Equation (7.17) and Antolin et al. [2019]; Johansson et al. [2019]) and the penalty method with penalization parameter $\beta = 10^2/h^p$ (cf. Equation (7.15)), where h denotes the mesh size, as explained in Section 7.5. Although the two methods yield different total fluxes, Figure 7.3b shows that the internal energy converges to the same value.

Moreover, if the numerical flux is integrated over Γ_0 and Γ_1, the obtained values are similar but not in perfect equilibrium. Such a difference is displayed in Figure 7.3c, where the relative error between the two fluxes is computed as

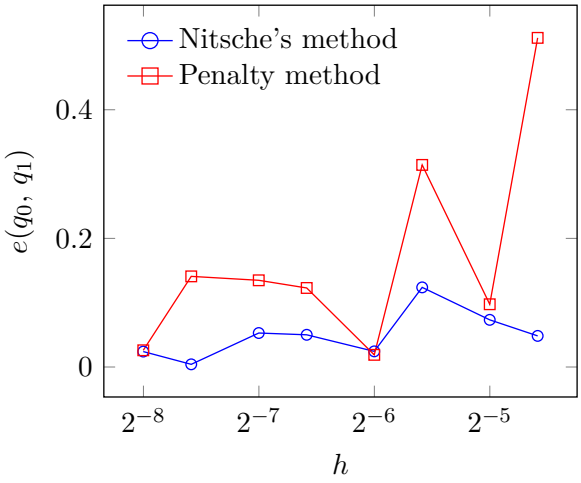
$$e(q_0^h, q_1^h) = \left| 1 - \frac{q_0^h}{-q_1^h} \right|, \tag{7.5}$$



(a) Total flux obtained by numerical integration.



(b) Internal energy.



(c) Equilibrium error (cf. Equation (7.5)).

Figure 7.3: Flux across the Dirichlet boundary Γ_1, internal energy, and equilibrium for a sequence of bisected meshes.

where q_0^h and q_1^h represent the integrated flux $\boldsymbol{\kappa} \nabla u^h \cdot \boldsymbol{n}$ defined by the numerical solution u^h over Γ_0 and Γ_1, respectively. Namely,

$$q_i^h = \int_{\Gamma_i} \boldsymbol{\kappa} \nabla u^h \cdot \boldsymbol{n} \, \mathrm{d}\Gamma, \qquad i \in \{0, 1\}. \tag{7.6}$$

This example indicates that integrating the numerical flux does not use all information contained in the finite element solution. Indeed, the underlying variational principle finds a solution that fulfills equilibrium in a global and local (element) sense [Bathe, 2007; Hughes et al., 2000]. Therefore, this information is contained in the solution, and the remaining of the chapter is devoted to the development of a strategy to accurately extract it.

This chapter lays out an explanation for computing the total flux and reaction forces

based on equilibrium considerations, generalizing the traditional approach to

- non-nodal shape functions that do not necessarily form a partition of unity,
- trimmed meshes.

Moreover, the proposed approach seems robust concerning the different methods for imposing the weak boundary conditions, in the sense that penalty and Nitsche's methods will converge to the same total-flux value.

7.3 The strong form of the model problem

Let $\Omega \subset \mathbb{R}^D$ be a bounded Lipschitz domain with disjoint Dirichlet and Neumann boundaries Γ_g, Γ_h, respectively, such that $\overline{\Gamma_g \cup \Gamma_h} = \partial\Omega$, $\Gamma_g \cap \Gamma_h = \emptyset$. The strong form of the heat conduction problem reads

$$-\nabla \cdot (\boldsymbol{\kappa}\, \nabla u) = f \qquad \text{in } \Omega, \tag{7.7}$$

$$u = g \qquad \text{on } \Gamma_g, \tag{7.8}$$

$$\boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n} = h \qquad \text{on } \Gamma_h, \tag{7.9}$$

where $\boldsymbol{\kappa} \in \mathbb{R}^{D\times D}$ is the thermal-conductivity tensor, $h : \Gamma_h \to \mathbb{R}$ is the prescribed flux, $g : \Gamma_g \to \mathbb{R}$ is the prescribed temperature, $f : \Omega \to \mathbb{R}$ is the volumetric heat supply, and $\boldsymbol{n} \in \mathbb{R}^D$ is the vector normal to the boundary.

7.4 The weak form for strong boundary conditions

Given the set of trial functions $\mathcal{S}_{g,\Gamma_g}(\Omega)$ and the test space $\mathcal{W}_{0,\Gamma_g}(\Omega)$,

$$\mathcal{S}_{g,\Gamma_g}(\Omega) = \{u \in \mathcal{H}^1(\Omega) \,|\, u = g \text{ on } \Gamma_g\}, \tag{7.10}$$

$$\mathcal{W}_{0,\Gamma_g}(\Omega) = \{w \in \mathcal{H}^1(\Omega) \,|\, w = 0 \text{ on } \Gamma_g\}, \tag{7.11}$$

the weak form of the problem reads

$$\begin{aligned} &\text{find } u \in \mathcal{S}_{g,\Gamma_g}(\Omega) \\ &\text{such that } a(w,\, u) = l(w) \qquad \forall w \in \mathcal{W}_{0,\Gamma_g}(\Omega). \end{aligned} \tag{W}$$

Here, $a(w,\, u)$ and $l(w)$ denote the classic bilinear and linear forms

$$a(w,\, u) = (\nabla w,\, \boldsymbol{\kappa}\nabla u)_\Omega, \tag{7.12}$$

$$l(w) = (w,\, f)_\Omega + (w,\, h)_{\Gamma_h}, \tag{7.13}$$

7.4.1 The Galerkin form

Problem (W) can be rewritten with homogeneous Dirichlet boundary conditions by lifting g to Ω. In particular, let $g_\Omega \in \mathcal{H}^1(\Omega)$ be such that $g_\Omega|_{\Gamma_g} = g$. Then, $u_0 = u - g_\Omega$ belongs to $\mathcal{W}_{0,\Gamma_g}(\Omega)$ and Problem (W) can be stated as

$$\begin{aligned} &\text{find } u_0 \in \mathcal{W}_{0,\Gamma_g}(\Omega) \\ &\text{such that } a(w,\, u_0) = l(w) - a(w,\, g_\Omega) \qquad \forall w \in \mathcal{W}_{0,\Gamma_g}(\Omega). \end{aligned}$$

The Galerkin form of Problem (W) with a finite-dimensional subspace $\mathcal{W}^h_{0,\Gamma_g}(\Omega) \subset \mathcal{W}_{0,\Gamma_g}(\Omega)$ and an approximation g^h to g_Ω reads

$$\begin{aligned} &\text{find } u^h \in \mathcal{W}^h_{0,\Gamma_g}(\Omega) \subset \mathcal{W}_{0,\Gamma_g}(\Omega), \\ &\text{such that } a(w^h, u^h) = l(w^h) - a(w^h, g^h), \qquad \forall w^h \in \mathcal{W}^h_{0,\Gamma_g}(\Omega). \end{aligned} \tag{G}$$

7.5 The weak form for weak boundary conditions

In case the temperature boundary conditions are applied weakly, these are not incorporated in the solution and test spaces. Instead, an additional term $a_w(\cdot, \cdot)$ associated with the energy of the constraint violation is added as follows

$$\begin{aligned} &\text{find } u \in \mathcal{H}^1(\Omega) \\ &\text{such that } a(w, u) + a_w(w, u) = l(w) \qquad \forall w \in \mathcal{H}^1(\Omega). \end{aligned} \tag{w}$$

The term $a_w(w, u)$ can assume different forms depending on the weak-boundary approach. For the penalty method [Babuška, 1973] with a penalty parameter $\beta \in \mathbb{R}$, $a_w(w, u) = a_\beta(w, u)$ will be defined as

$$a_\beta(w, u) = (w, \beta(u - g))_{\Gamma_{\tilde{g}}}. \tag{7.14}$$

Typically, when using finite-element shape functions of polynomial order p, β is a mesh-dependent parameter scaled with h^p to retain the expected convergence rates [Utku and Carey, 1982]

$$\beta = \bar{\beta}\frac{1}{h^p}, \tag{7.15}$$

where $\bar{\beta} \in \mathbb{R}$ is a user-specified parameter, often dependent on the material parameters.

For the symmetric Nitsche's method [Nitsche, 1971] with stabilization parameter $\gamma \in \mathbb{R}$, $a_w(w, u) = a_\gamma(w, u)$ is defined as

$$a_\gamma(w, u) = -(\boldsymbol{\kappa}\nabla w \cdot \boldsymbol{n}, u - g)_{\Gamma_g} - (w, \boldsymbol{\kappa}\nabla u \cdot \boldsymbol{n})_{\Gamma_g} + (w, \gamma(u - \tilde{g}))_{\Gamma_g}. \tag{7.16}$$

In this work, γ is scaled as in the original publication Nitsche [1971]

$$\gamma = \bar{\gamma}\frac{1}{h}. \tag{7.17}$$

For immersed methods, better estimates for γ can be obtained by solving a global or element-local generalized eigenvalue problem (cf., e.g., de Prenter et al. [2018]; Griebel and Schweitzer [2003]). Similar estimates through generalized eigenvalue problems are also employed for variationally-consistent patch coupling (cf., e.g., Apostolatos et al. [2014]; Embar et al. [2010]; Hansbo [2005]; Harari and Grosu [2015]; Hu et al. [2018]; Jiang et al. [2015]; Nguyena et al. [2013]; Ruess et al. [2014]).

7.6 The trimmed-domain Galerkin form

Following Section 6.1, the trimmed Galerkin form is defined through a simple domain Ω^{fict} containing the physical domain $\Omega \subset \Omega^{\text{fict}}$ and a finite-dimensional subspace $\mathcal{W}^h(\Omega^{\text{fict}}) \subset \mathcal{H}^1(\Omega^{\text{fict}})$. The trimmed Galerkin form of Problem (w) can be formulated as

$$\begin{aligned} &\text{find } u^h \in \mathcal{W}^h(\Omega), \\ &\text{such that } a(w^h, u^h) + a_w(w^h, u^h) = l(w^h), \qquad \forall w^h \in \mathcal{W}^h(\Omega)., \end{aligned} \tag{g}$$

where

$$\mathcal{W}^h(\Omega) = span\left\{ w^h\big|_\Omega \,:\, w^h \in \mathcal{W}^h(\Omega^{\text{fict}}) \right\}. \tag{7.18}$$

7.7 Conservative reactions to strong boundary conditions

In this section, the traditional way to compute the reactions is interpreted as testing a weak problem with specific test functions. This point of view will allow generalizing the computation of the reactions to trimmed domains and to bases that do not form a partition of unity. This interpretation is inspired by Brezzi et al. [2001]; Hughes et al. [2000] and similar to the argumentation therein. However, in this work, the focus is on obtaining the (integrated) total reaction flux instead of a "pointwise" approximation of the normal flux by a function defined on the boundary.

A conservative way to compute the reactions can be derived by considering a problem compatible with the mixed problem in Equations (7.7)–(7.9). Namely, other than the temperature boundary condition $u = g$ on Γ_g, the compatible reaction flux r is assumed to exist and is prescribed on Γ_g. The flux r is such that the condition $u = g$ is retained on Γ_g. The remaining data of the problem $\boldsymbol{\kappa}$, f, and h are unchanged. For elastic problems, this corresponds to prescribing the forces that would enforce the displacement conditions.

In particular, let us consider the following boundary-value problem with compatible conditions on Γ_g

$$-\nabla \cdot (\boldsymbol{\kappa}\, \nabla u) = f \qquad \text{in } \Omega, \tag{7.19}$$

$$u = g \qquad \text{on } \Gamma_g, \tag{7.20}$$

$$\boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n} = r \qquad \text{on } \Gamma_g, \tag{7.21}$$

$$\boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n} = h \qquad \text{on } \Gamma_h. \tag{7.22}$$

For simplicity, the data $\boldsymbol{\kappa}$, f, r, g, h, and the boundary $\partial\Omega$ are assumed to be "smooth enough" for the following manipulations to hold. Given a solution $u^* \in \mathcal{H}^2(\Omega)$ for the mixed problem of Equations (7.7)–(7.9), it will also be a solution for the problem of Equations (7.19)–(7.22) with $r = (\boldsymbol{\kappa}\, \nabla u^* \cdot \boldsymbol{n})|_{\Gamma_g}$. Indeed, u^* satisfies Equations (7.19), (7.20), and (7.22), as they are the same as Equations (7.7)–(7.9). Moreover, Equation (7.21) is trivially satisfied by the definition $r = (\boldsymbol{\kappa}\, \nabla u^* \cdot \boldsymbol{n})|_{\Gamma_g}$.

Following standard variational arguments, one can formulate a weak form by multiplying Equation (7.19) by a test function w belonging to a test space chosen to be $\mathcal{W} = \mathcal{H}^1(\Omega)$ and integrating over Ω. This yields the following weak form

$$\begin{aligned} &\text{find } u \in \mathcal{S}_{g,\Gamma_g}(\Omega), \\ &\text{such that } a(w,\, u) = l(w) + (w,\, r)_{\Gamma_g}, \qquad \forall w \in \mathcal{W} = \mathcal{H}^1(\Omega). \end{aligned} \tag{R}$$

Note that the test space consists of the whole $\mathcal{H}^1(\Omega)$ function space, not requiring the test functions to be zero on any part of the boundary. In particular, the boundedness of Ω ensures that the constant $w = 1$ belongs to the test space $\mathcal{W} = \mathcal{H}^1(\Omega)$. Testing Problem (R) with $w = 1$ assures global equilibrium

$$0 = \int_{\Omega} f \,\mathrm{d}\boldsymbol{x} + \int_{\Gamma_h} h \,\mathrm{d}\boldsymbol{x} + \int_{\Gamma_g} r \,\mathrm{d}\boldsymbol{x}. \tag{7.23}$$

A solution $u^* \in \mathcal{H}^2(\Omega)$ for the original weak Problem (W) will also solve the strong form in Equations (7.19)–(7.22) and the compatible Problem (R).

Moreover, since $\mathcal{W}_{0,\Gamma_g}(\Omega)$ is a closed subspace of $\mathcal{H}^1(\Omega)$, then $\mathcal{H}^1(\Omega)$ admits the direct-sum representation [Rudin, 1991; Salsa, 2016]

$$\mathcal{H}^1(\Omega) = \mathcal{W}_{0,\Gamma_g}(\Omega) \oplus \mathcal{W}_{0,\Gamma_g}(\Omega)^{\perp}.$$

Namely, each $w \in \mathcal{H}^1(\Omega)$ admits a (unique) representation $w_0 + w_g$, with $w_0 \in \mathcal{W}_{0,\Gamma_g}(\Omega)$ and $w_g \in \mathcal{W}_{0,\Gamma_g}(\Omega)^{\perp}$. Following Bazilevs and Hughes [2007]; Hughes [2000]; Hughes et al. [2000], the arbitrarity of $w_0 + w_g = w \in \mathcal{H}^1(\Omega)$ in Problem (R) implies the arbitrarity of w_0 and w_g, allowing to reformulate the problem as

$$\begin{aligned} &\text{find } u \in \mathcal{S}_{g,\Gamma_g}(\Omega) \\ &\text{such that } a(w_0,\, u) = l(w_0), \qquad \forall w_0 \in \mathcal{W}_{0,\Gamma_g}(\Omega), \quad \text{and} \qquad (7.24) \\ &\qquad\qquad a(w_g,\, u) = l(w_g) + (w_g,\, r)_{\Gamma_g}, \qquad \forall w_g \in \mathcal{W}_{0,\Gamma_g}(\Omega)^{\perp}. \qquad (7.25) \end{aligned}$$

Equation (7.24) is precisely the original variational form for strong boundary conditions in Problem (W). Therefore, if the compatible weak Problem (R) has a solution, this will also be the solution of the original Problem (W). Assuming the latter problem to have a unique solution in $\mathcal{S}_{g,\Gamma_g}(\Omega)$, this will identify the solution to the former problem.

Consequently, given an appropriate reaction flux r that makes the variational form in Problem (R) compatible with the original weak form in Problem (W), the conventional way to compute the reactions for conforming meshes can be interpreted as testing the variational form in Problem (R) with appropriate test functions. The total flux computed in this way will naturally satisfy the equilibrium expression given by the variational form. Specifically,

1. given a solution $u^* \in \mathcal{S}_{g,\Gamma_g}(\Omega)$ for the original weak Problem (W),
2. assume that there exists an $r \in \mathcal{L}_2(\Gamma_g)$ such that the variational form in Problem (R) holds for $u = u^*$.

3. Then, the unknown total flux $\int_{\Gamma_g} r \, \mathrm{d}\Gamma$ is obtained by testing the compatible variational form in Problem (R) with a function $w_g \in \mathcal{H}^1$ such that $w_g|_{\Gamma_g} = 1$.
4. The obtained total flux $\int_{\Gamma_g} r \, \mathrm{d}\Gamma$ will be in global equilibrium with the other fluxes, as the compatible variational form in Problem (R) also holds for $w = 1 \in \mathcal{H}^1$, yielding the global equilibrium in the sense of Equation (7.23).

Indeed, inserting w_g in Problem (R) yields

$$\begin{aligned} \int_{\Gamma_g} r \, \mathrm{d}\Gamma &= (w_g, \, r)_{\Gamma_g} \\ &= a(w_g, \, u^*) - l(w_g), \end{aligned} \tag{7.26}$$

where the term $a(w_g, \, u^*) - l(w_g)$ can be evaluated for known w_g and u^*.

The test function w_g defines the linear functional $R_{w_g}(u)$ associated with the reactions and defined as

$$R_{w_g}(u) = a(w_g, \, u) - l(w_g). \tag{7.27}$$

Note that such a functional is defined not only when $r \in \mathcal{L}_2(\Gamma_g)$, but it is continuous for any $u \in \mathcal{H}^1(\Omega)$, and l belongs to $\mathcal{H}^1(\Omega)^*$, the dual space of $\mathcal{H}^1(\Omega)$.

Similarly, the reactions on multiple disjoint Dirichlet boundaries $\{\Gamma_g^i\}_{i=1\ldots n_b}$, such that

$$\Gamma_g = \bigcup_{i=1}^{n_b} \Gamma_g^i, \tag{7.28}$$

can be computed by means of test functions w_g^i such that $w_g^i|_{\Gamma_g^i} = 1$, $w_g^i|_{\Gamma_g^j} = 0$ for $i \neq j$.

7.7.1 Reactions for the Galerkin form

Employing the classical nodal finite element method (cf., e.g., Bathe [2007]; Hughes [2000]; Hughes et al. [2000]; Strang [1973]), the space $\mathcal{W}^h_{0,\Gamma_g}(\Omega)$ in the Galerkin Problem (G) is commonly based on a discretization that partitions Ω into a finite number of elements $\{\Omega_e\}_{e=1..n_e}$

$$\overline{\Omega} = \bigcup_{e=1}^{n_e} \overline{\Omega}_e.$$

Following Hughes [2000]; Hughes et al. [2000], let $\eta = \{1, 2, \ldots, n_d\}$ be the set of indices of the associated nodes $\mathcal{N} = \{\boldsymbol{x}_A\}_{A\in\eta} \subset \overline{\Omega}$ and $\eta_g = \{A \, : \, \boldsymbol{x}_A \in \Gamma_g\} \subset \eta$ be the subset containing indices of nodes lying on Γ_g. Given the linear-independent nodal shape functions $\{N_A\}_{A\in\eta}$, where N_A is associated with node $\boldsymbol{x}_A$, the space spanned by $\{N_A\}_{A\in\eta}$ admits the direct-sum decomposition

$$\mathrm{span}\{N_A\}_{A\in\eta} = \underbrace{\mathrm{span}\{N_A\}_{A\in\eta\setminus\eta_g}}_{\mathcal{W}^h_{0,\Gamma_g}(\Omega)} \oplus \mathrm{span}\{N_A\}_{A\in\eta_g}. \tag{7.29}$$

The functions $\{N_A\}_{A\in\eta\setminus\eta_g}$ are a basis for the space $\mathcal{W}^h_{0,\Gamma_g}(\Omega)$, while $\{N_A\}_{A\in\eta_g}$ are commonly used to define g^h

$$g^h = \sum_{A\in\eta_g} g^h_A N_A. \tag{7.30}$$

The discrete linear system of equations takes the form

$$\boldsymbol{K}\boldsymbol{d} = \boldsymbol{F}, \tag{7.31}$$

where

$$\begin{aligned} K_{AB} &= a(N_A,\, N_B), && A, B \in \eta, \\ F_A &= l(N_A), && A \in \eta. \end{aligned}$$

Equation (7.31) can be partitioned into the blocks associated with the nodes identified by $\eta \setminus \eta_g$ and η_g

$$\left[\begin{array}{c|c} \boldsymbol{K}_{00} & \boldsymbol{K}_{0g} \\ \hline \boldsymbol{K}_{0g}^\top & \boldsymbol{K}_{gg} \end{array}\right] \left[\begin{array}{c} \boldsymbol{d}_0 \\ \hline \boldsymbol{d}_g \end{array}\right] = \left[\begin{array}{c} \boldsymbol{F}_0 \\ \hline \boldsymbol{F}_g \end{array}\right],$$

where

$$\begin{aligned} [\boldsymbol{K}_{00}]_{AB} &= a(N_A,\, N_B), && A, B \in \eta \setminus \eta_g, \\ [\boldsymbol{K}_{0g}]_{AB} &= a(N_A,\, N_B), && A \in \eta \setminus \eta_g, B \in \eta_g, \\ [\boldsymbol{K}_{gg}]_{AB} &= a(N_A,\, N_B), && A, B \in \eta_g. \end{aligned}$$

The upper blocks yield the traditional problem for $\boldsymbol{d}_0$ with strong boundary conditions corresponding to Problem (G)

$$\boldsymbol{K}_{00}\boldsymbol{d}_0 = \boldsymbol{F}_0 - \boldsymbol{K}_{0g}\boldsymbol{d}_g. \tag{7.32}$$

The lower blocks correspond to the nodal forces associated with the reactions.

The computation of the reactions viewed as testing the variational form as in Equation (7.26), corresponds in the discrete case to testing the Galerkin form in Problem (G) with a $w^h_g \in \operatorname{span}\{N_A\}_{A\in\eta_g}$ such that $w^h_g\big|_{\Gamma_g} = 1$. For the discrete matrix system of equations, this corresponds to a left-multiplication by a coefficient vector representing the coordinates of w^h_g in the basis $\{N_A\}_{A\in\eta_g}$. In the case of the considered nodal partition-of-unity basis $\{N_A\}$, this takes the form

$$\left[0...0 \;\middle|\; 1...1\right] \left\{ \left[\begin{array}{c|c} \boldsymbol{K}_{00} & \boldsymbol{K}_{0g} \\ \hline \boldsymbol{K}_{0g}^\top & \boldsymbol{K}_{gg} \end{array}\right] \left[\begin{array}{c} \boldsymbol{d}_0 \\ \hline \boldsymbol{d}_g \end{array}\right] - \left[\begin{array}{c} \boldsymbol{F}_0 \\ \hline \boldsymbol{F}_g \end{array}\right] \right\} = \left[0...0 \;\middle|\; 1...1\right] \left[\begin{array}{c} \boldsymbol{0} \\ \hline \boldsymbol{r} \end{array}\right], \tag{7.33}$$

where the top block vanishes, as $\boldsymbol{d}_0$ solves Equation (7.32), and $\boldsymbol{r}$ represents the nodal reactions. Similarly, given a boundary portion $\Gamma_0 \subset \Gamma_g$, if it is possible to construct a test function $w^h_{g,0} \in \operatorname{span}\{N_A\}_{A\in\eta_g}$ such that $w^h_{g,0}\big|_{\Gamma_0} = 1$ and $w^h_{g,0}\big|_{\Gamma_g\setminus\Gamma_0} = 0$, then the reaction can be obtained by multiplication with a vector composed of the coordinates $w^h_{g,0}$ in the basis $\{N_A\}_{A\in\eta_g}$. This corresponds to the traditional algorithm in Table 7.1, as summarized in Table 7.2.

Continuous	(Equation (7.26))	$a(w_g,\, u) - l(w_g)$
Discrete	(Equation (7.33))	$\left[\begin{array}{c\|c} 0\ldots0 & 1\ldots1 \end{array}\right] \left\{ \left[\begin{array}{c\|c} \boldsymbol{K}_{00} & \boldsymbol{K}_{0g} \\ \hline \boldsymbol{K}_{0g}^\top & \boldsymbol{K}_{gg} \end{array}\right] \left[\begin{array}{c} \boldsymbol{d}_0 \\ \hline \boldsymbol{d}_g \end{array}\right] - \left[\begin{array}{c} \boldsymbol{F}_0 \\ \hline \boldsymbol{F}_g \end{array}\right] \right\}$
Algorithm	(Table 7.1)	$\sum_{A \in \eta_g} \int_\Omega \nabla N_A \cdot \left(\boldsymbol{\kappa} \nabla u^h\right) - N_A f \,\mathrm{d}\Omega - \int_{\Gamma_h} N_A h \,\mathrm{d}\Gamma$

Table 7.2: Traditional algorithm to compute the reactions viewed as testing the weak and Galerkin form with a specific test function.

7.8 Conservative reactions for trimmed meshes

Interpreting the total reaction as testing the weak form with specific test functions serves as a basis to obtain total conservative reactions on trimmed meshes. In the case of weak boundary conditions, the test space in Problem (w) naturally consists of the whole $\mathcal{H}^1(\Omega)$, containing elements w such that $w|_{\Gamma_g} = 1$. Therefore, it is not necessary to consider a compatible problem including the reactions. Instead, motivated by the principle of virtual work in Problem (w), the weak boundary condition term represents the normal flux action on the test functions with trace on Γ_g. In particular, given a $w_g \in \mathcal{H}^1$ such that $w_g|_{\Gamma_g} = 1$, the total flux can be computed by evaluating either side of

$$a(w_g,\, u) - l(w_g) = -a_w(w_g,\, u). \tag{7.34}$$

Note that the total flux computed as in Equation (7.34) is in the form of the extraction expressions studied in Babuška and Miller [1984]. For the Nitsche's method in Equation (7.16), this is further supported by the fact that it is variationally consistent. Namely, assuming enough regularity, integrating by parts, and using the arbitrarity of the test functions, the original strong form in Equations (7.7)–(7.9) is recovered. Therefore, a weak solution $u^* \in \mathcal{H}^2$ for Problem (w), with the weak boundary-condition term as in Equation (7.16), will also solve both the compatible strong form in Equations (7.19)–(7.22) with $r = (\boldsymbol{\kappa}\, \nabla u^* \cdot \boldsymbol{n})|_{\Gamma_g}$ and the associated weak form in Problem (R). The reactions can be computed as in Equation (7.26).

For the penalty method [Babuška, 1973], the weak form in Problem (w), with the weak boundary-condition term as in Equation (7.14), corresponds to the following perturbed strong form

$$-\nabla \cdot (\boldsymbol{\kappa}\, \nabla u) = f \qquad \text{in } \Omega, \tag{7.35}$$

$$\boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n} + \beta(u - g) = 0 \qquad \text{on } \Gamma_g, \tag{7.36}$$

$$\boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n} = h \qquad \text{on } \Gamma_h. \tag{7.37}$$

From Equation (7.36), it follows that $(1,\, \boldsymbol{\kappa}\, \nabla u \cdot \boldsymbol{n})_{\Gamma_g} = -(1,\, \beta(u-g))_{\Gamma_g} = -a_\beta(w_g,\, u) = a(w_g,\, u) - l(w_g)$ is a natural approximation to the flux on Γ_g.

Given the shape functions $\{N_A\}$, let $\tilde{\eta}(\Gamma_0) = \{A : N_A|_{\Gamma_0} \neq 0\}$ be the set of indices of shape functions with non-zero trace on Γ_0. It is assumed $N_A|_{\Gamma_g \setminus \Gamma_0} = 0 \;\; \forall A \in \tilde{\eta}(\Gamma_0)$.

1. For each $A \in \tilde{\eta}(\Gamma_0)$, compute the discrete fluxes

$$q_A = \int_\Omega \nabla N_A \cdot \left(\boldsymbol{\kappa} \nabla u^h\right) \, \mathrm{d}\Omega, \qquad q_A^e = \int_\Omega N_A f \, \mathrm{d}\Omega - \int_{\Gamma_h} N_A h \, \mathrm{d}\Gamma.$$

2. The reaction r on Γ_0 is obtained by summing the fluxes of shape functions with non-zero trace on Γ_0

$$r = \sum_{A \in \tilde{\eta}(\Gamma_0)} q_A - q_A^e.$$

Table 7.3: Algorithm for computing the reactions on trimmed meshes with partition-of-unity shape functions.

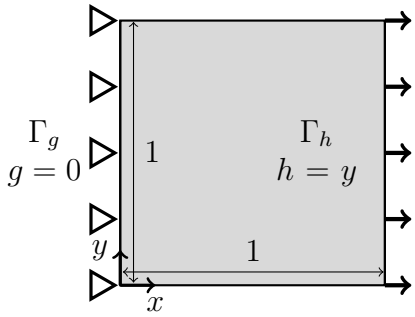
7.8.1 Reactions for the Galerkin form

In order to compute the total flux on a disjoint portion of the boundary $\Gamma_0 \subset \Gamma_g$ for partition-of-unity bases on trimmed domains, one strategy can be to define a function $w^h \in \mathcal{W}^h(\Omega)$ that is one in a neighborhood of Γ_0, and has zero trace on $\Gamma_g \setminus \Gamma_0$. In particular, the function w^h, such that $w^h\big|_{\Omega_e} = 1$ for each element Ω_e cut by Γ_0, will also be such that $w^h\big|_{\Gamma_0} = 1$, even for a complex boundary Γ_0 that cannot be interpolated exactly by the shape functions.

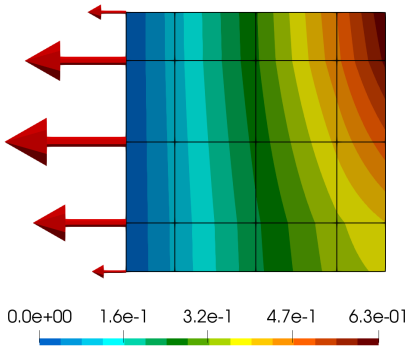
Algorithmically, the only necessary modification to the procedure in Table 7.1 is to sum the fluxes q_A associated with functions N_A *with non-zero trace* on Γ_0 and zero trace on $\Gamma_g \setminus \Gamma_0$. An example is shown in Figure 7.4, where standard reactions on nodal linear shape functions are visually compared to the trimmed-mesh reactions with linear and quadratic B-splines shape functions (cf. Hughes et al. [2005]). Note that in Figure 7.4c the first two columns of control points are needed to compute the reactions, as these are the linear functions with support on the constrained boundary. In Figure 7.4d the first three columns of control points have to be considered for computing the reaction, as the basis functions' support grows with the order. This procedure can be summarized as in Table 7.3.

7.9 Conservative reactions for bases not forming a partition of unity

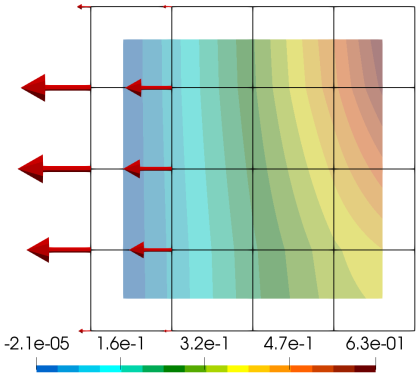
Equations (7.26) and (7.34) are already in a general form, suitable for bases that do not form a partition of unity. Using the same ideas as in Section 7.8, the strategy is to define a test function w^h that is one on each cut element. With the reasonable assumption that the basis functions $\{N_A\}$ can represent constants, let $c_A \in \mathbb{R}$ be the coefficient associated



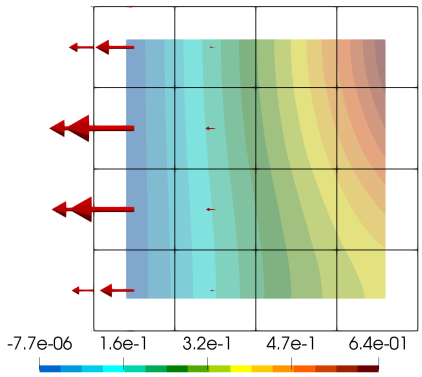
(a) Geometry description.



(b) Strong boundary conditions, linear elements.



(c) Nitsche's method, linear elements, stabilization parameter $\gamma = 100$.



(d) Nitsche's method, quadratic B-splines, stabilization parameter $\gamma = 100$.

Figure 7.4: Solution field, mesh, and reactions for trimmed meshes. The reactions are depicted as red arrows in the x-direction located at the control points.

with the shape function N_A, such that

$$\sum_A c_A N_A = 1 \qquad \text{on } \overline{\Omega}. \tag{7.38}$$

The computation of the reactions can be summarized as in Table 7.4, where the sum in Table 7.3 is generalized to a *weighted sum* of fluxes associated with basis functions with non-zero trace on Γ_0. Note that for partition-of-unity bases, it holds $c_A = 1$ for any A. In this case, the procedure in Table 7.4 is the same as the one in Table 7.3.

7.9.1 Reactions for hierarchical B-splines

For hierarchical B-splines, the coefficients $\{c_A\}$ can be obtained by projecting onto the hierarchical mesh the coefficients representing the function one on the base level. Since the standard B-splines form a partition of unity [Piegl and Tiller, 1995], the base-level

Given the shape functions $\{N_A\}$, let $\tilde{\eta}(\Gamma_0) = \{A \,|\, N_A|_{\Gamma_0} \neq 0\}$ be the set of indices of shape functions with non-zero trace on Γ_0. It is assumed $N_A|_{\Gamma_g \setminus \Gamma_0} = 0 \;\; \forall A \in \tilde{\eta}(\Gamma_0)$. Let $\{c_A\} \subset \mathbb{R}$ be the coordinates of 1 in the basis $\{N_A\}$, as in Equation (7.38).

1. For each $A \in \tilde{\eta}(\Gamma_0)$, compute the discrete fluxes

$$q_A = \int_\Omega \nabla N_A \cdot \left(\boldsymbol{\kappa} \nabla u^h\right) \, \mathrm{d}\Omega, \qquad q_A^e = \int_\Omega N_A f \, \mathrm{d}\Omega - \int_{\Gamma_h} N_A h \, \mathrm{d}\Gamma$$

2. The reaction r on Γ_0 is obtained by a weighted sum of fluxes associated with shape functions with non-zero trace on Γ_0

$$r = \sum_{A \in \tilde{\eta}(\Gamma_0)} c_A \, (q_A - q_A^e).$$

Table 7.4: Algorithm for computing the reactions on trimmed meshes. The basis functions do not need to form a partition of unity.

coefficients are all equal to one. Let $\boldsymbol{c}^e$ be the vector of coefficients $\{c_A\}$ associated with functions with support on the element Ω_e, then $\boldsymbol{c}^e$ can be obtained as follows

$$\boldsymbol{c}^e = \boldsymbol{C}^e \mathbf{1}, \tag{7.39}$$

where $\mathbf{1}$ is a vector of ones and $\boldsymbol{C}^e$ is the element hierarchical extraction operator (see Chapter 4 and Lorenzo et al. [2017]; Scott et al. [2014]). Algorithmically, this projection can be performed as described in Chapter 5. See Figure 7.5a for an example of values for the coefficients $\{C_A\}$.

7.9.2 Reactions for integrated Legendre polynomials

The basis functions used in the p-version of the finite element method do not form a partition of unity. Given an order p, such univariate basis is defined in the interval $[-1, 1]$ as [Szabó and Babuška, 1991]

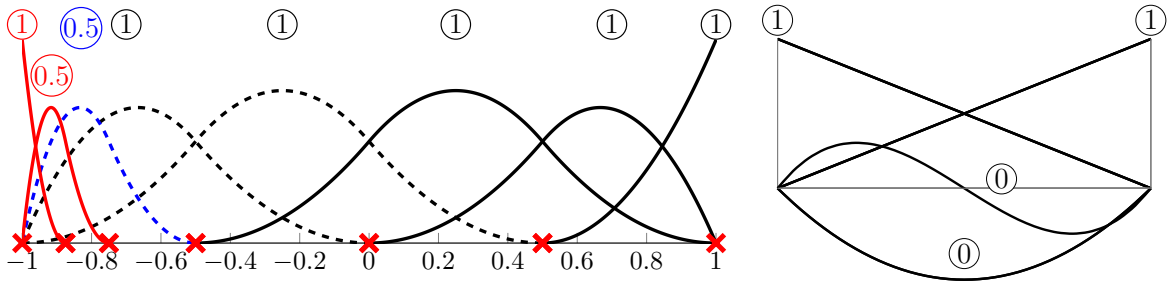
$$\hat{\xi}_1(r) = \frac{1}{2}(1 + r) \tag{7.40}$$

$$\hat{\xi}_2(r) = \frac{1}{2}(1 - r) \tag{7.41}$$

$$\hat{\xi}_i(r) = P_{i-1}(r) \qquad i = 2, 3, \ldots, p+1, \tag{7.42}$$

where $\hat{\xi}_1(r)$ and $\hat{\xi}_2(r)$ are the classical linear shape functions, while P_{i-1} is defined by an integral expression of the Legendre polynomials L_i

$$P_i(r) = \sqrt{\frac{2i-1}{2}} \int_{-1}^{r} L_{i-1}(t) \, \mathrm{d}t = \frac{1}{\sqrt{4i-2}} \left(L_i(r) - L_{i-2}(r)\right) \qquad i = 2, 3, \ldots \, .$$



(a) Coefficients for a hierarchical B-spline basis.

(b) Coefficients for the integrated-Legendre basis [Szabó and Babuška, 1991] of order $p = 3$.

Figure 7.5: Example of coefficients (circled numbers) for computing the reactions with bases that do not form a partition of unity.

Since the linear shape functions form a partition of unity $\hat{\xi}_1 + \hat{\xi}_2 = 1$ on $[-1, 1]$, the remaining high-order functions $\hat{\xi}_i$, $i \geq 3$, will have a zero coefficient. See Figure 7.5b for an example. Similarly, for a basis obtained by the tensor product of the univariate basis in Equations (7.40)–(7.42), the coefficients will be the tensor product of the univariate coefficients. Namely, the linear shape functions will have coefficient one, while the remaining high-order functions will have a zero coefficient. This section agrees with the extraction of nodal forces presented in Babuška and Miller [1984]; Szabó and Babuška [2011].

7.10 2D benchmark

In this section, a smooth problem involving a flux induced by a temperature difference on a curved geometry is considered. In two dimensions, a simple benchmark can be formulated on a quarter of annulus Ω with inner and outer radii r_1 and r_2, respectively (cf. Figure 7.6a). In particular, let us consider

$$\begin{aligned} -\nabla \cdot (\boldsymbol{\kappa} \nabla u) &= 0 && \text{in } \Omega = \left\{ \boldsymbol{x} \in (0, r_2)^2 : r_1 < \|\boldsymbol{x}\| < r_2 \right\}, \\ u &= 2 \ln (r_1) && \text{on } \Gamma_0 = \{ \boldsymbol{x} \in \partial\Omega : \|\boldsymbol{x}\| = r_1 \}, \\ u &= 2 \ln (r_2) && \text{on } \Gamma_1 = \{ \boldsymbol{x} \in \partial\Omega : \|\boldsymbol{x}\| = r_2 \}, \\ \boldsymbol{\kappa} \nabla u \cdot \boldsymbol{n} &= 0 && \text{on } \partial\Omega \setminus (\overline{\Gamma_0 \cup \Gamma_1}), \end{aligned}$$

where $\boldsymbol{\kappa}$ is the identity matrix. The analytical solution of the problem is the harmonic function (cf. Figures 7.6b and 7.6c)

$$u = 2 \ln \|\boldsymbol{x}\|.$$

Note that the data of the problem do not specify any external flux. The global equilibrium only assures that the flux across the Dirichlet boundary Γ_0 balances the flux across Γ_1. However, such total flux cannot be obtained directly from the source term or the boundary conditions.

The domain Ω is immersed in a Cartesian mesh of the bounding box $\Omega^{\text{fict}} = (0, r_2)^2$. One mesh example is shown in Figure 7.6d, along with the reparameterized integration-domains used to resolve the discontinuity in the integrands, as explained in Kudela et al. [2015, 2016]. The problem is solved with both Nitsche's and penalty methods, as in Equations (7.14) and (7.16), with parameters $\bar{\beta} = 10^2$ (cf. Equation (7.15)) and $\bar{\gamma} = 10(p+1)^2$ (cf. Equation (7.17)), similarly to Antolin et al. [2019]; Johansson et al. [2019]. The immersed analysis is compared to the solution obtained by a conforming NURBS mesh with similar element size h and strong Dirichlet boundary conditions.

The energy error of the numerical solution u^h is computed with respect to the bilinear form $a(\cdot, \cdot)$ of the original problem without weak boundary conditions. In particular, the error

$$e(u^h) = \sqrt{\frac{1}{2}\, a(u - u^h, u - u^h)} \tag{7.43}$$

for the conforming mesh is shown in Figure 7.7a to have a similar convergence behavior for both Nitsche's and penalty methods. The conservative fluxes q_0^c, q_1^c are computed on the boundaries Γ_0 and Γ_1 according to Table 7.3. The direct fluxes are numerically integrated as follows

$$q_i^h = \int_{\Gamma_i} \boldsymbol{\kappa} \nabla u^h \cdot \boldsymbol{n} \, \mathrm{d}\Gamma, \qquad i \in \{0, 1\}. \tag{7.44}$$

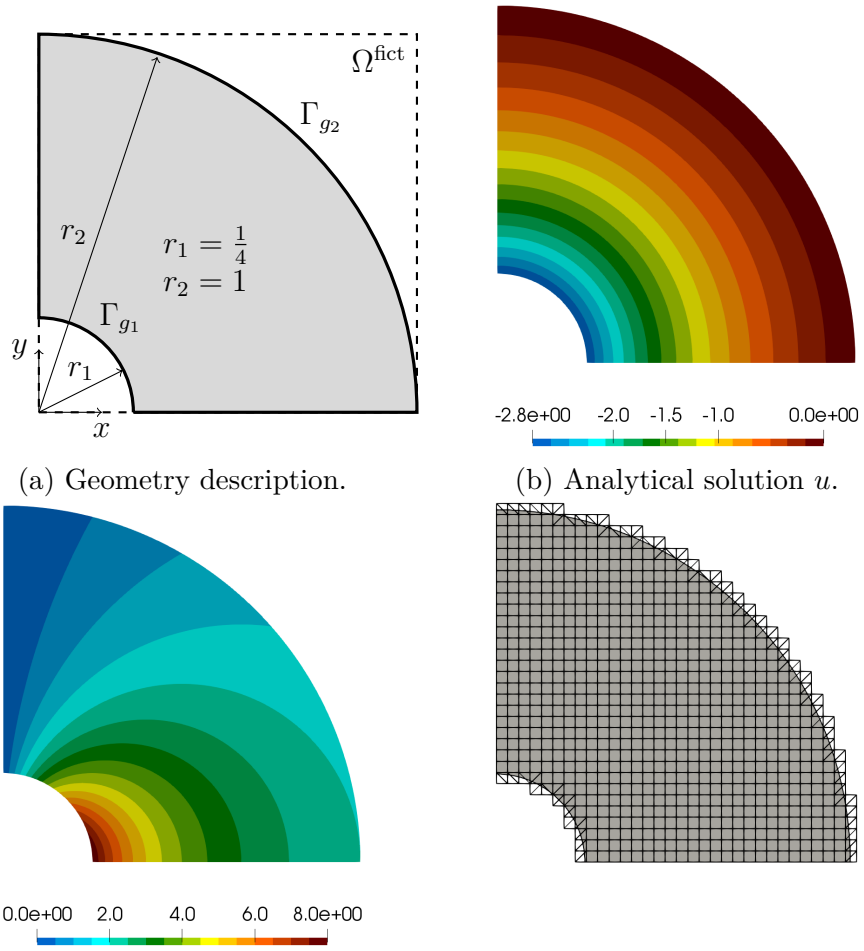
Figures 7.7c and 7.7d show the relative flux error

$$e_i(q) = \left| 1 - \frac{q}{\int_{\Gamma_i} \boldsymbol{\kappa} \nabla u \cdot \boldsymbol{n} \, \mathrm{d}\Gamma} \right|. \tag{7.45}$$

for both the direct fluxes $e_i(q^h)$ (dashed lines) and for the conservative ones $e_i(q^c)$ (solid lines). Note that the conservative reactions yield more accurate results than the direct approach and show an apparent convergence to the analytical total flux. Nitsche's method yields convergence rates that are two times higher than the strain-energy error rates, similarly to those obtained with the conforming mesh (cf. Figures 7.7b and 7.7d). This phenomenon is often referred to as superconvergence [Babuška and Miller, 1984; Hughes et al., 2000; Szabó and Babuška, 2011; Wahlbin, 1995]. These rates of convergence are not attained by the penalty method, as shown in Figure 7.7c. Indeed, the penalty method accurately computes the reactions of a perturbed problem, and the penalty parameter is scaled with h^p. Instead, if the penalty parameter is scaled as $\beta = \bar{\beta}/h^{2p}$, the same rates of convergence as the conforming mesh and Nitsche's method are attained, as shown in Figure 7.8a.

The equilibrium error

$$e(q_0, q_1) = \left| 1 - \frac{q_0}{-q_1} \right| \tag{7.46}$$



(a) Geometry description.

(b) Analytical solution u.

(c) x-component $u_{,x}$ of the analytical flux.

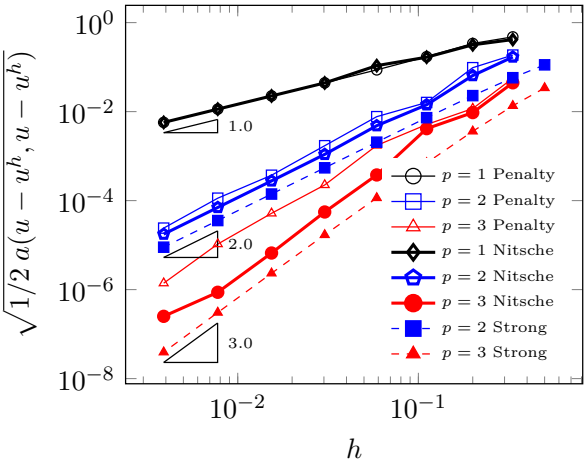
(d) Example of computational mesh. The diagonal lines show the integration domains [Kudela et al., 2015].

Figure 7.6: 2D benchmark. Geometry, analytical solution, and mesh example.

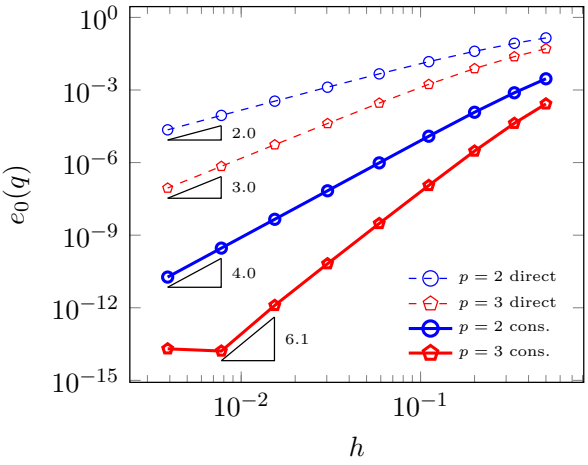
is shown in Figure 7.8b for both direct fluxes $e(q_0^h, q_1^h)$ (dashed lines) and conservative fluxes $e(q_0^c, q_1^c)$ (solid lines). Note that the conservative fluxes are in equilibrium up to small numerical inaccuracies that grow as the condition number with order $O(h^{-2})$. The direct-flux equilibrium error is several orders of magnitude higher than the one for the conservative fluxes.

7.11 Façade element

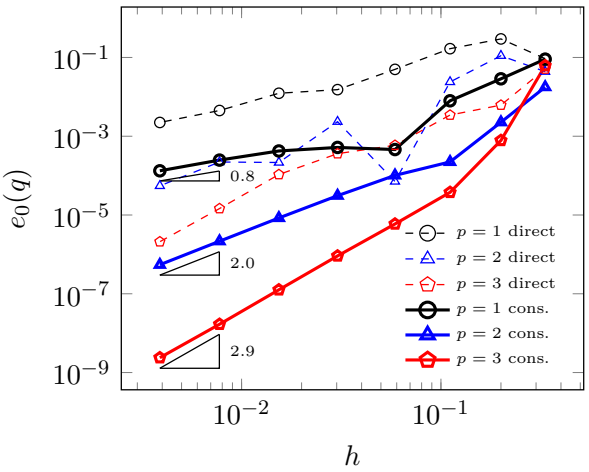
The conservative reactions applied to the model problem for the façade element introduced in Section 7.2 yield a total flux converging to the same value for both Nitsche's and penalty methods. This behavior does not seem to hold for the direct approach: compare Figures 7.9a and 7.9b to Figures 7.3a and 7.3c.



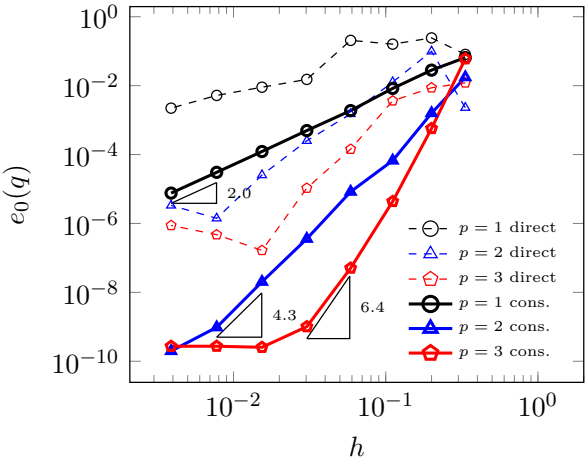
(a) Energy error



(b) Strong boundary conditions: total flux error.



(c) Penalty method: total flux error.



(d) Nitsche's method: total flux error.

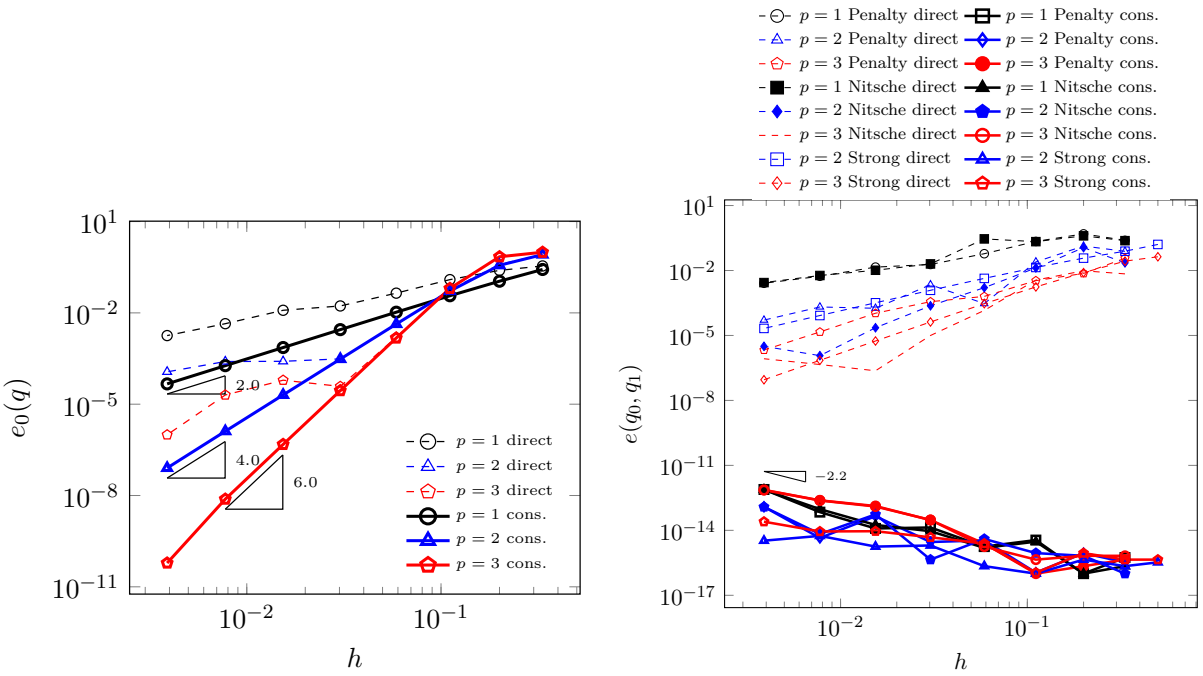
Figure 7.7: 2D benchmark. Energy error and flux errors for direct fluxes (dashed lines) and conservative fluxes (solid lines).

7.12 Trimmed Kirchhoff-Love shell example

The presented reaction computation can be extended to the weak form of the Kirchhoff-Love shell problem with weak boundary conditions introduced in Section 6.2.

In the following, the parameter $\bar{\beta}$ in Equations (6.12) and (6.13) is chosen as $\bar{\beta} = 10^3$, since in Herrema et al. [2019] this value is shown to be suitable for various examples in the context of multi-patch penalty coupling.

Following the reasoning of the previous sections, the ith reaction component, r_i, corresponding to the traction on Γ_g is computed by testing the variational form with a test function $\boldsymbol{w}^{g,i} \in \mathcal{H}^2$ such that $w_i^{g,i}\big|_{\Gamma_g} = 1$, $w_j^{g,i}\big|_{\Gamma_g} = 0$ for $j \neq i$ and such that $(\Phi(\boldsymbol{w}^{g,i}) \cdot \boldsymbol{n})\big|_{\Gamma_\theta} = 0$. In particular, given a known displacement field $\boldsymbol{u}^* \in \mathcal{H}^2$, the ith



(a) Penalty method: total flux error with penalty parameter $\beta = \bar{\beta}/h^{2p}$.

(b) Equilibrium error of direct fluxes (dashed lines) and conservative fluxes (solid lines). The conservative fluxes (lines below) are in equilibrium up to machine precision.

Figure 7.8: 2D benchmark. Equilibrium error and improved convergence in the flux error obtained by the penalty method.

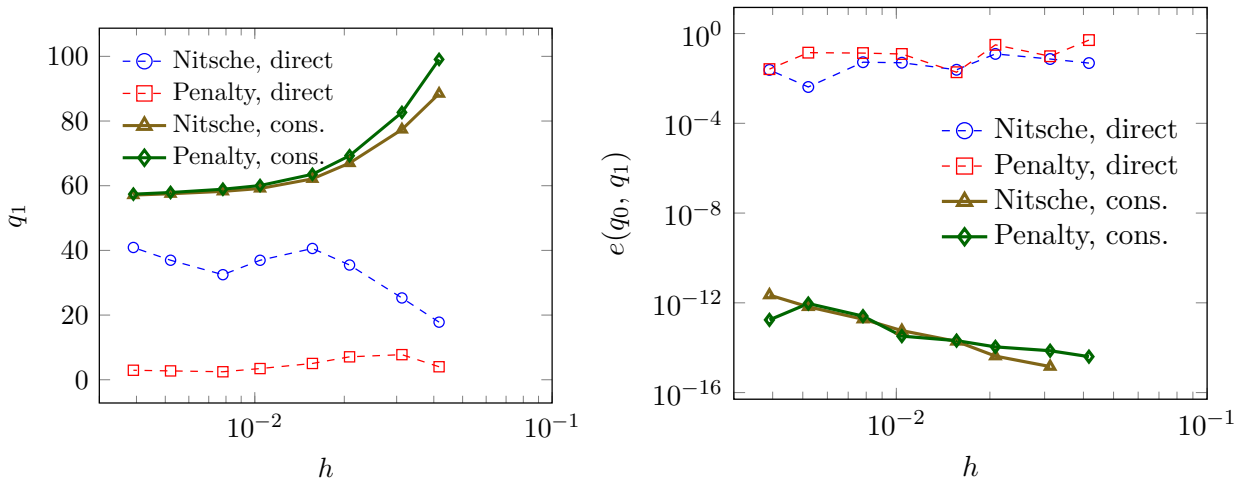
component of the total reaction can be computed by evaluating either side of the following equation

$$a(\boldsymbol{w}^{g,i}, \boldsymbol{u}^*) - l(\boldsymbol{w}^{g,i}) = -b^{\text{disp}}(\boldsymbol{w}^{g,i}, \boldsymbol{u}^*). \tag{7.47}$$

The same strategy as in Sections 7.8 and 7.9 can be applied to the present case to evaluate the reactions on trimmed geometries with bases that do not form a partition of unity. The global equilibrium is confirmed in the example shown in Figure 7.10a. The edges of the circular hole on the left (blue curves) are clamped, while a traction $\boldsymbol{t} = (0, 0, 1)^\top$ is applied on the straight boundary marked in Figure 7.10a (red arrows).

The geometry is described by a B-spline patch stored in a STandard for the Exchange of Product model data (STEP) file format [ISO 10303-11:1994, 1994]. A (trimmed) computational mesh is obtained for numerical analysis by k-refinement on the geometric patch, as described in Cottrell et al. [2009]; Hughes et al. [2005]. The STEP file also contains the trimming curves in the parametric space of the B-spline patch, allowing to define accurate shell integration rules following Kudela et al. [2015], as explained in Coradello et al. [2020b]; Rank et al. [2011].

The problem is solved with an initial (trimmed) B-spline patch of uniform degree $p = 3$. The elements intersecting the physical domain Ω are shown in Figure 7.10a. Figure 7.10b



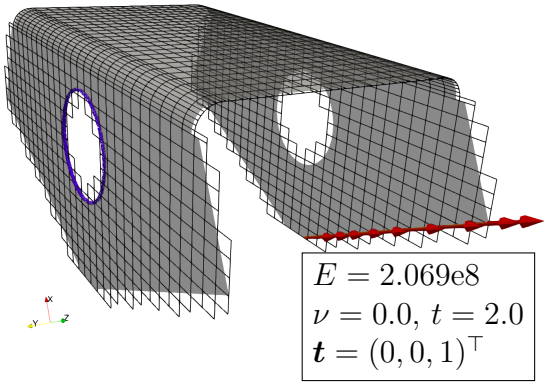
(a) Total flux. (b) Equilibrium error.

Figure 7.9: Façade element example. Total flux and equilibrium error for the direct fluxes (dashed lines) and conservative fluxes (solid lines).

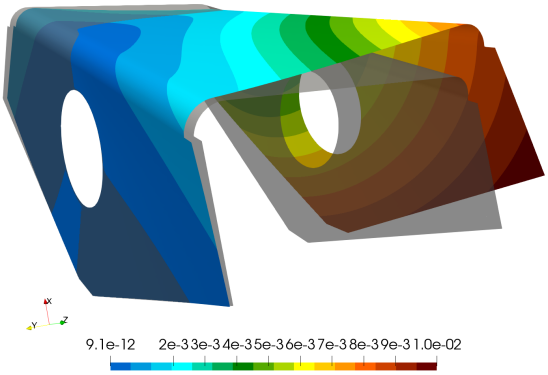
shows the displacement magnitude on the deformed geometry. The problem is also solved with hierarchical B-splines with several refinement levels. The elements cut by the clamped boundary are recursively refined up to a refinement level l. Additionally, some elements totally outside the physical domain are refined to ensure that the finest-level hierarchical functions are activated, as explained in Coradello et al. [2020a]. Specifically, for each cut-element Ω_e marked for refinement, it is also marked for refinement each element $\tilde{\Omega}_e \in \Omega^{\text{fict}} \setminus \overline{\Omega}$ contained in the support of basis functions of element Ω_e. See Coradello et al. [2020a] for details. A graded mesh is obtained by enforcing a mesh-admissibility class equal to one [Bracco et al., 2019; Buffa and Giannelli, 2016]. Namely, each element can have active basis functions belonging to at most two consecutive levels. Details can be found in Bracco et al. [2019]; Buffa and Giannelli [2016]; Carraturo et al. [2019]. Figure 7.10c shows the mesh obtained after $l = 5$ recursive refinements, along with the von Mises stress around the clamped hole.

The basis functions having non-zero trace on the clamped edge belong to the hierarchical-refinement levels l and $l-1$. These functions do not form a partition of unity, and the reaction tractions are computed as described in Section 7.9.1. The mesh and discrete reactions for $l \in \{0, 2, 5\}$ are shown in Figures 7.10e–7.10g. Figure 7.10d shows the relative equilibrium error of the reaction traction $\boldsymbol{r}$ on the clamped edge with the applied external traction $\boldsymbol{t}$ computed as follows

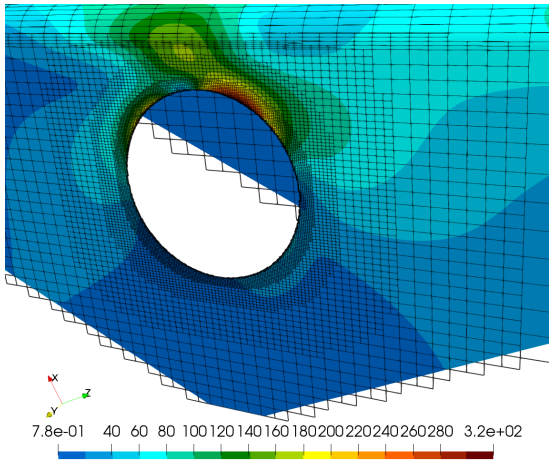
$$e(\boldsymbol{r}, \boldsymbol{t}) = \frac{\|\boldsymbol{r} - \boldsymbol{t}\|_2}{\|\boldsymbol{t}\|_2}. \tag{7.48}$$



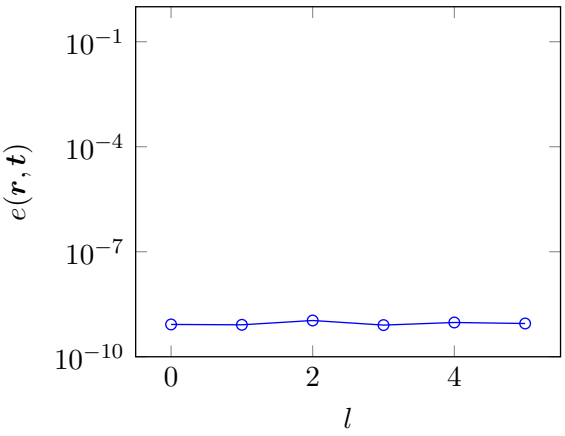
(a) Initial mesh, and boundary conditions: clamped edges (blue curves) and distributed traction (red arrows).



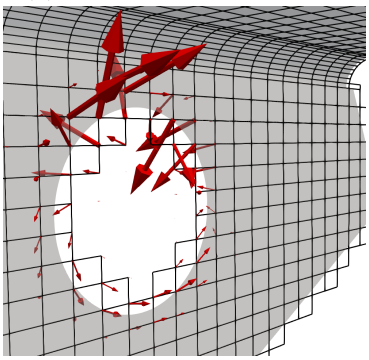
(b) Displacement magnitude.



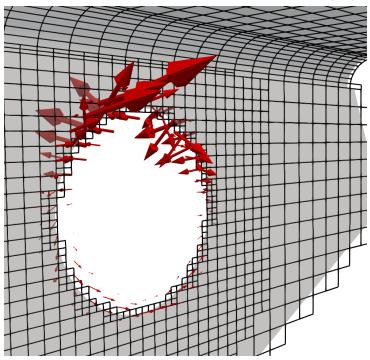
(c) Von Mises stress around clamped hole.



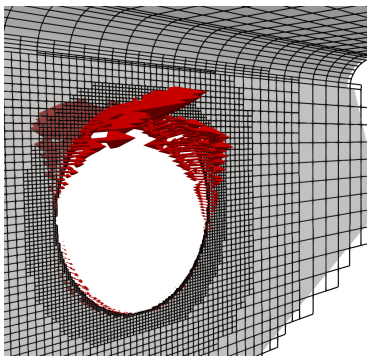
(d) Equilibrium error.



(e) Reaction tractions on initial mesh.



(f) Reaction tractions with 2 refinement levels.



(g) Reaction tractions with 5 refinement levels.

Figure 7.10: Trimmed Kirchhoff-Love shell example.

Chapter 8

Matrix-free approach to locally-refined finite cell analyses

Chapter 5 introduced some algorithms to extract the (truncated) hierarchical B-splines and project the DOFs during mesh refinement and coarsening. In this chapter, these algorithms are combined to multiply a vector by the finite element system matrix, formulating a matrix-free framework combining

- local refinement,
- trimming through the finite cell method, and
- sum factorization.

8.1 Introduction

In the context of solving a linear system of equations $\boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}$, the term "matrix-free" refers to an iterative solution procedure accessing matrix $\boldsymbol{K}$ only through matrix-vector products $\boldsymbol{K}\boldsymbol{v}$ for different search directions $\boldsymbol{v}$. The product $\boldsymbol{K}\boldsymbol{v}$ is computed implicitly without storing matrix $\boldsymbol{K}$ explicitly. This approach becomes of particular advantage, or even necessary, for large systems whose explicit storage is too expensive in memory requirements.

Consider, for example, the conjugate-gradient iterative solver given in Algorithm 8.1. The only use of matrix $\boldsymbol{K}$ is to compute the matrix-vector products $\boldsymbol{K}\boldsymbol{x}_0$ and $\boldsymbol{K}\boldsymbol{p}_j$. Therefore, if it is possible to obtain $\boldsymbol{K}\boldsymbol{v}$ without storing $\boldsymbol{K}$ in memory, its storage can be spared altogether.

In finite element analysis, the system matrix $\boldsymbol{K}$ is generally obtained by assembling individual element matrices $\boldsymbol{K}^e$

$$\boldsymbol{K} = \sum_e (\boldsymbol{A}^e)^\top \boldsymbol{K}^e \, \boldsymbol{A}^e, \tag{8.1}$$

where $\boldsymbol{A}^e$ are zero-one assembly matrices that map global degrees of freedom to element-local degrees of freedom. Equation (8.1) allows to compute the matrix-vector product

Algorithm 8.1: Conjugate gradient algorithm to solve $\boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}$ starting from an initial solution guess $\boldsymbol{x}_0$ [Saad, 2003].

$\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{x}_0$
$\boldsymbol{p}_0 = \boldsymbol{r}_0$
$j = 0$
Loop
  $\alpha_j = \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(\boldsymbol{K}\boldsymbol{p}_j, \boldsymbol{p}_j)}$
  $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$
  $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j \boldsymbol{K}\boldsymbol{p}_j$
  if $\boldsymbol{r}_{j+1}$ is sufficiently small **then**
    exit loop
  end
  $\beta_j = \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}$
  $\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$
  $j = j + 1$
End Loop

$\boldsymbol{K}\boldsymbol{v}$ (without explicitly storing $\boldsymbol{K}$) as

$$\begin{aligned} \boldsymbol{K}\boldsymbol{v} &= \sum_e (\boldsymbol{A}^e)^\top \boldsymbol{K}^e \, \boldsymbol{A}^e \boldsymbol{v} \\ &= \sum_e (\boldsymbol{A}^e)^\top \boldsymbol{K}^e \, \boldsymbol{v}^e, \end{aligned} \tag{8.2}$$

where $\boldsymbol{v}^e = \boldsymbol{A}^e \boldsymbol{v}$. Storing all element matrices $\boldsymbol{K}^e$ usually is not a beneficial solution, as it would require more memory than an explicit sparse representation of $\boldsymbol{K}$. The disadvantage is particularly pronounced for isogeometric analysis and high-continuity approaches, as the location maps of neighboring elements present significant overlaps. Alternatively, the element matrices $\boldsymbol{K}^e$ could be recomputed on the fly. However, this can result in very high run-time. A further expansion of Equation (8.2) in terms of the quadrature of $\boldsymbol{K}^e$ allows for a more efficient hybrid approach, as shown in the next section.

8.2 Sum factorization

The element matrix $\boldsymbol{K}^e$ is typically obtained by numerical integration, i.e., as a weighted sum

$$\boldsymbol{K}^e \approx \sum_{q=1}^{n_q} \boldsymbol{K}^e(\boldsymbol{\xi}_q) \, w_q$$

over the integration points $\boldsymbol{\xi}_q$ and weights w_q. In the remaining of this chapter, the superscript e is dropped to favor readability.

Assuming that

Assumption 1 the integration points $\boldsymbol{\xi}_q$ have a tensor-product structure, and
Assumption 2 the shape functions have a tensor-product structure,

typical finite-element matrices will exhibit a specific structure that can be leveraged to obtain efficient algorithms. These assumptions will be formally defined later in this section. A strategy using these properties was first introduced with the name of sum factorization for spectral finite elements in Orszag [1980]. Successively, this approach was applied to high-order C^0 finite elements (see, e.g., Melenk et al. [2001]) and isogeometric analysis (see, e.g., Antolin et al. [2015]). In this work, an element-by-element assembly is considered, favoring standard finite element implementations. Higher efficiency can be achieved when sum factorization is combined with weighted quadrature for the whole element patch, as presented in Calabrò et al. [2019]; Calabrò et al. [2017]; Hiemstra et al. [2019]; Sangalli and Tani [2018]. However, weighted quadrature for trimming and local refinement is still an active area of research.

The sum factorization is first exemplified in a matrix form for the mass matrix, following closely Sangalli and Tani [2018] and their notation. Consider the element parameter space $\hat{\Omega} = [0,1]^D$. Given a parametric direction $d \in \{1, ..., D\}$, the ${}^d n_f$ univariate element basis functions are denoted by ${}^d\hat{N}_i(\xi)$, $i = 0...{}^d n_f - 1$. For a multi-index $\boldsymbol{i} = (i_1, ..., i_D)$, Assumption 2 means that element basis functions $\hat{N}_{\boldsymbol{i}}(\boldsymbol{\xi})$ at $\boldsymbol{\xi} = (\xi_1, ..., \xi_D)^\top$ admit the decomposition

$$\hat{N}_{\boldsymbol{i}}(\boldsymbol{\xi}) = {}^1\hat{N}_{i_1}(\xi_1) \; ... \; {}^D\hat{N}_{i_D}(\xi_D).$$

Assumption 1 means that the integration points are arranged in a Cartesian grid. Namely, given the ${}^d n_q$ univariate locations ${}^d\xi_q$, $q = 0...{}^d n_q - 1$, the full set of integration points can be written as

$$\boldsymbol{\xi}_{\boldsymbol{q}} = ({}^1\xi_{q_1}, ..., {}^D\xi_{q_D})^\top. \tag{8.3}$$

In the following, the multi-index $\boldsymbol{i}$ will be often identified by the scalar index i according to the usual tensor-product order

$$i = i_1 + {}^1n_f(i_2 + {}^2n_f(...i_{D-1} + {}^{D-1}n_f \; i_D)).$$

Following the isoparametric paradigm, the element domain Ω is obtained from a mapping $\boldsymbol{F}$

$$\Omega = \boldsymbol{F}(\hat{\Omega}), \text{ with } \boldsymbol{F} = \sum_{\boldsymbol{i}} \hat{N}_{\boldsymbol{i}} \, \boldsymbol{P}_{\boldsymbol{i}},$$

for some control points $\boldsymbol{P}_{\boldsymbol{i}}$. The global basis functions are therefore defined as

$$N_{\boldsymbol{i}} = \hat{N}_{\boldsymbol{i}} \circ \boldsymbol{F}^{-1}.$$

The Galerkin mass matrix $\boldsymbol{M}$ is defined as

$$M_{ij} = \int_{\Omega} \rho(\boldsymbol{x}) \; N_i(\boldsymbol{x}) \; N_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \tag{8.4}$$

$$= \int_{\hat{\Omega}} g(\boldsymbol{\xi}) \; \hat{N}_i(\boldsymbol{\xi}) \; \hat{N}_j(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \tag{8.5}$$

Algorithm 8.2: Mass matrix-vector multiplication (element-local version of the algorithm presented in Sangalli and Tani [2018]).

Input: ${}^d\boldsymbol{B} \in \mathbb{R}^{{}^d n_q \times {}^d n_f}, d = 1 \dots D$
$\boldsymbol{G} \in \mathbb{R}^{n_q \times n_q}$
$\boldsymbol{v} \in \mathbb{R}^{n_f}$
Output: $\boldsymbol{r} = \tilde{\boldsymbol{M}}\boldsymbol{v} = \left({}^D\boldsymbol{B}^\top \otimes \dots \otimes {}^1\boldsymbol{B}^\top\right) \boldsymbol{G} \left({}^D\boldsymbol{B} \otimes \dots \otimes {}^1\boldsymbol{B}\right) \boldsymbol{v}$.

1 $\boldsymbol{w}_1 = \left({}^D\boldsymbol{B} \otimes \dots \otimes {}^1\boldsymbol{B}\right) \boldsymbol{v}$ `// using Algorithm 5.7 or 5.8`
2 $\boldsymbol{w}_2 = \boldsymbol{G}\boldsymbol{w}_1$ `// straightforward, as` $\boldsymbol{G}$ `is diagonal`
3 $\boldsymbol{r} = \left({}^D\boldsymbol{B}^\top \otimes \dots \otimes {}^1\boldsymbol{B}^\top\right) \boldsymbol{w}_2$ `// using Algorithm 5.7 or 5.8`

where $\rho(\boldsymbol{x})$ is the material density, $g(\boldsymbol{\xi}) = \det(\boldsymbol{J})\, \rho(\boldsymbol{x}(\boldsymbol{\xi}))$, and $\boldsymbol{J}$ is the Jacobian of $\boldsymbol{F}$. Numerical integration over the quadrature points $\boldsymbol{\xi}_q$ and weights w_q, $q = 1 \dots n_q$, yields

$$M_{ij} \approx \tilde{M}_{ij} = \sum_{q=1}^{n_q} g(\boldsymbol{\xi}_q)\ \hat{N}_i(\boldsymbol{\xi}_q)\ \hat{N}_j(\boldsymbol{\xi}_q)\, w_q. \tag{8.6}$$

The tensor structure of the quadrature is best shown in matrix form. Consider the matrix $\boldsymbol{B}$, $B_{qj} = \hat{N}_j(\boldsymbol{\xi}_q)$, and the diagonal matrix $\boldsymbol{G}$, $G_{qq} = g(\boldsymbol{\xi}_q) w_q$. Equation (8.6) can be formulated as

$$\tilde{\boldsymbol{M}} = \boldsymbol{B}^\top \boldsymbol{G} \boldsymbol{B}. \tag{8.7}$$

When both Assumption 1 and Assumption 2 are fulfilled, the tensor structure is transferred to $\boldsymbol{B}$, giving

$$\boldsymbol{B} = \left({}^D\boldsymbol{B}^\top \otimes \dots \otimes {}^1\boldsymbol{B}^\top\right), \tag{8.8}$$

where ${}^d B_{qj} = {}^d\hat{N}_j({}^d\xi_q)$ and ${}^d\xi_q$, $q = 1 \dots {}^d n_q$ are the integration points in the dth parametric direction. Therefore, Equation (8.7) becomes

$$\tilde{\boldsymbol{M}} = \left({}^D\boldsymbol{B}^\top \otimes \dots \otimes {}^1\boldsymbol{B}^\top\right) \boldsymbol{G} \left({}^D\boldsymbol{B} \otimes \dots \otimes {}^1\boldsymbol{B}\right). \tag{8.9}$$

This structure reveals that the multiplication $\tilde{\boldsymbol{M}}\boldsymbol{v}$ can be performed through Algorithms 5.7 and 5.8 introduced in Section 5.4.1, as outlined in Algorithm 8.2. The number of multiplications in Algorithm 8.2 when using $\hat{n}_f$ basis functions and $\hat{n}_q$ integration points in each parametric direction can be verified to be

$$2\left(\hat{n}_q \hat{n}_f^D + \hat{n}_q^2 \hat{n}_f^{D-1} + \dots + \hat{n}_q^D \hat{n}_f\right) + \hat{n}_q^D. \tag{8.10}$$

When $\hat{n}_f \neq \hat{n}_q$, the cost can be written as

$$2\hat{n}_f \hat{n}_q \frac{\hat{n}_f^D - \hat{n}_q^D}{\hat{n}_f - \hat{n}_q} + \hat{n}_q^D. \tag{8.11}$$

8.3 Sum factorization for local refinement

Unfortunately, Assumption 2 is generally not satisfied when using local refinement. For some bases, each function can be written as a product of univariate functions. For example, this fact holds for hierarchical B-splines (see Section 3.3) or the multi-level *hp*-basis presented in Zander [2017]; Zander et al. [2016, 2015, 2017]. However, the element basis is not a full tensor-product of univariate functions. For some other bases, even single basis functions are not a product of univariate functions. See, e.g., the truncated hierarchical B-splines introduced in Section 3.4. In both cases, Equation (8.7) cannot be written as Equation (8.9), inhibiting the use of Algorithm 8.2.

The multi-level extraction operator introduced in Chapter 4 relates the (global or element-local) locally-refined basis to a reference space with standard basis functions that can be chosen to form a full tensor product. In this case, after a vector $\boldsymbol{v}$ is transformed to the reference space, the sum factorization can be applied precisely as before. Afterward, the result has to be transformed back to the space spanned by the locally-refined basis.

In particular, recalling relations Equations (4.8) and (4.9), the element extraction operator $\boldsymbol{C}$ can map some reference basis functions $\{s_j(\boldsymbol{\xi})\}_{j=1..n_s}$ (as the Bernstein polynomials) to the locally-refined hierarchical basis $\{h_j(\boldsymbol{\xi})\}_{j=1..n_h}$. Similarly, its transpose can translate the coefficients with respect to the basis $\{h_j(\boldsymbol{\xi})\}$ to the coefficients with respect to the basis $\{s_j(\boldsymbol{\xi})\}$. Let $\boldsymbol{S}$ and $\boldsymbol{H}$ be the matrices composed of standard and locally-refined functions collocated at the integration points, respectively. Specifically, let $S_{qj} = s_j(\boldsymbol{\xi}_q)$ and $H_{qj} = h_j(\boldsymbol{\xi}_q)$. Operator $\boldsymbol{C}$ relates $\boldsymbol{H}$ and $\boldsymbol{S}$ as follows

$$\boldsymbol{H} = \boldsymbol{S}\,\boldsymbol{C}^\top. \tag{8.12}$$

Assuming Assumption 1 and Assumption 2 hold for the standard functions $\{s_j(\boldsymbol{\xi})\}$ and integration points $\{\boldsymbol{\xi}_q\}$, $\boldsymbol{S}$ admits the following tensor decomposition analogous to Equation (8.8)

$$\boldsymbol{S} = \left({}^D\boldsymbol{S}^\top \otimes \ldots \otimes {}^1\boldsymbol{S}^\top\right). \tag{8.13}$$

The multiplication $\tilde{\boldsymbol{M}}\boldsymbol{v}$ can be written as

$$\begin{aligned}\tilde{\boldsymbol{M}}\boldsymbol{v} &= \boldsymbol{H}^\top\boldsymbol{G}\boldsymbol{H}\boldsymbol{v}\\ &= \boldsymbol{C}\boldsymbol{S}^\top\boldsymbol{G}\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{v}\\ &= \boldsymbol{C}\left({}^D\boldsymbol{S}^\top \otimes \ldots \otimes {}^1\boldsymbol{S}^\top\right)\boldsymbol{G}\left({}^D\boldsymbol{S} \otimes \ldots \otimes {}^1\boldsymbol{S}\right)\boldsymbol{C}^\top\boldsymbol{v}.\end{aligned} \tag{8.14}$$

Equation (8.14) can be evaluated exploiting the tensor product, as shown in Algorithm 8.3 for (truncated) hierarchical B-splines. The transformation from Bernstein polynomials ${}^d\boldsymbol{S}$ to B-splines ${}^d\boldsymbol{N}$ can be done in each parametric direction separately (${}^d\boldsymbol{N} = {}^d\boldsymbol{E}\,{}^d\boldsymbol{S}$, $d = 1...D$). The cost for $\hat{n}_s$ basis functions and $\hat{n}_q$ integration points in each parametric direction reads (see Equation (8.10), Algorithms 5.10 and 5.13)

$$2(n_l - 1)D\hat{n}_s^{D+1} + Dn_s^2 n_q + 2\left(\hat{n}_q\hat{n}_s^D + \hat{n}_q^2\hat{n}_s^{D-1} + \ldots + \hat{n}_q^D\hat{n}_s\right) + \hat{n}_q^D, \tag{8.15}$$

where n_l is the number of refinement levels having at least one active function.

Algorithm 8.3: Mass matrix-vector multiplication for locally refined bases, according to Equation (8.14).

1 $\boldsymbol{w}_1 = \boldsymbol{C}^\top \boldsymbol{v}$ // using Algorithm 5.13
2 $\boldsymbol{w}_2 = \left({}^D\boldsymbol{S} \otimes \ldots \otimes {}^1\boldsymbol{S}\right) \boldsymbol{w}_1$ // using Algorithm 5.7 or 5.8
3 $\boldsymbol{w}_3 = \boldsymbol{G}\boldsymbol{w}_2$ // straightforward, as $\boldsymbol{G}$ is diagonal
4 $\boldsymbol{w}_4 = \left({}^D\boldsymbol{S}^\top \otimes \ldots \otimes {}^1\boldsymbol{S}^\top\right) \boldsymbol{w}_3$ // using Algorithm 5.7 or 5.8
5 $\boldsymbol{r} = \boldsymbol{C}\boldsymbol{w}_4$ // using Algorithm 5.10

8.4 Sum factorization for trimmed meshes

As introduced in Section 6.1, accurate computations on trimmed meshes require non-standard integration rules for cut elements (see., e.g., Abedian et al. [2013a]; Breitenberger et al. [2015]; Hubrich et al. [2017]; Joulaian et al. [2016]; Kudela et al. [2015, 2016]; Marussig and Hughes [2018]; Müller et al. [2013]; Parvizian et al. [2007]; Rank et al. [2012]). See Marussig and Hughes [2018] for an extensive review. Many of these approaches define quadrature rules based on locations that are not arranged in a Cartesian grid, violating Assumption 1 and precluding the tensor structure in Equation (8.9). Examples of these integration rules are adaptive quadratures based on quadtrees or octrees (see, e.g., Parvizian et al. [2007]; Rank et al. [2012]), and boundary-conforming integration subdomain methods (see, e.g., Breitenberger et al. [2015]; Kudela et al. [2015, 2016]).

8.4.1 Moment fitting

The moment fitting equations can be used to define quadrature rules with locations arranged in a Cartesian grid [Müller et al., 2013].

In particular, given the integrands $\{\hat{f}_j\}_{j=1\ldots m}$, the moment fitting approach aims at constructing a quadrature rule over a domain $\hat{\Omega}$ with n integration points $\{\boldsymbol{\xi}_i\}_{i=1\ldots n}$ and weights $\{w_i\}_{i=1\ldots n}$. The approach is based on the moment fitting equations

$$\sum_i^n \hat{f}_j(\boldsymbol{\xi}_i)\, w_i = \int_{\hat{\Omega}} \hat{f}_j(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi}, \qquad j = 1\ldots m. \tag{8.16}$$

In matrix notation, Equation (8.16) reads

$$\begin{bmatrix} \hat{f}_1(\boldsymbol{\xi}_1) & \hat{f}_1(\boldsymbol{\xi}_2) & \ldots & \hat{f}_1(\boldsymbol{\xi}_n) \\ \hat{f}_2(\boldsymbol{\xi}_1) & \hat{f}_2(\boldsymbol{\xi}_2) & \ldots & \hat{f}_2(\boldsymbol{\xi}_n) \\ \vdots & \vdots & & \vdots \\ \hat{f}_m(\boldsymbol{\xi}_1) & \hat{f}_m(\boldsymbol{\xi}_2) & \ldots & \hat{f}_m(\boldsymbol{\xi}_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_{\hat{\Omega}} \hat{f}_1(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi} \\ \int_{\hat{\Omega}} \hat{f}_2(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi} \\ \vdots \\ \int_{\hat{\Omega}} \hat{f}_m(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi} \end{bmatrix}. \tag{8.17}$$

It is possible to solve Equation (8.17) for both $\{\boldsymbol{\xi}_i\}_{i=1...n}$ and weights $\{w_i\}_{i=1...n}$, resulting in a non-linear problem. In this work, the integration points $\{\boldsymbol{\xi}_i\}_{i=1...n}$ are fixed at the locations of the standard Gauss-Legendre quadrature rule, as in Hubrich et al. [2017]; Joulaian et al. [2016], yielding a linear problem for the unknown weights $\{w_i\}_{i=1...n}$. For polynomial integrands, a smart choice for the functions $\{\hat{f}_j\}_{j=1...m}$ is the Lagrange polynomials with nodes at the integration points $\{\boldsymbol{\xi}_i\}_{i=1...n}$ with $m = n$ [Hubrich and Düster, 2019]. In particular, let the points $\{\boldsymbol{\xi}_{\boldsymbol{i}}\}$, with multi-index $\boldsymbol{i} = (i_1, ..., i_D)$, $1 \leq i_d \leq {}^d n_q$, be the Cartesian product of the univariate integration points $\{{}^d\xi_i\}_{i=1...{}^d n_q}$ in the parametric directions $d = 1...D$. The univariate Lagrange polynomials $\{{}^d\ell_j(\xi)\}_{j=1...{}^d n_q}$ in each parametric direction d are defined on the nodes $\{{}^d\xi_i\}_{i=1...{}^d n_q}$ as

$$ {}^d\ell_j(\xi) = \prod_{\substack{i=1 \\ i \neq j}}^{{}^d n_q} \frac{\xi - {}^d\xi_i}{{}^d\xi_j - {}^d\xi_i}. \tag{8.18} $$

The integrands are defined as the tensor product Lagrange polynomials

$$ \hat{f}_{\boldsymbol{j}}(\boldsymbol{\xi}) = \ell_{\boldsymbol{j}}(\boldsymbol{\xi}) = {}^1\ell_{j_1}(\xi_1)...{}^D\ell_{j_D}(\xi_D). \tag{8.19} $$

The interpolation property of the chosen basis

$$ \ell_{\boldsymbol{j}}(\boldsymbol{\xi}_{\boldsymbol{i}}) = \begin{cases} 1 & \text{if } \boldsymbol{i} = \boldsymbol{j}, \\ 0 & \text{otherwise,} \end{cases} $$

makes the matrix in Equation (8.17) the identity, yielding the direct solution

$$ \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_{\hat{\Omega}} \ell_1(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \\ \int_{\hat{\Omega}} \ell_2(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \\ \vdots \\ \int_{\hat{\Omega}} \ell_n(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \end{bmatrix}. \tag{8.20} $$

The integrals on the right-hand side of Equation (8.20) can be computed through surface integrals [Hubrich et al., 2017; Joulaian et al., 2016; Müller et al., 2013] or any volumetric quadrature rule defined on locations not necessarily following a Cartesian structure. Note that the drawback of this approach is that to integrate functions of order $2p$ (as the entries of a mass matrix), $2p+1$ integration points per parametric directions are necessary (instead of the $p+1$ points needed by the standard Gaussian quadrature). Moreover, the quadrature weights do not form a tensor product.

8.4.2 Trimmed moment fitting

According to the introduction to trimmed analyses of Section 6.1, the physical geometry Ω is immersed into a fictitious domain Ω^{fict}, $\Omega \subset \Omega^{\text{fict}}$, typically of a topology that is

trivial to mesh (e.g., the smallest bounding box containing the physical domain Ω, or the shape of an untrimmed patch coming from a computer-aided design (CAD) file). The finite element mesh is defined on Ω^{fict}, and each element domain can contain parts of Ω^{fict} and Ω. Given the (discontinuous) domain-indicator function $\alpha(\boldsymbol{x}) : \Omega^{\text{fict}} \to [0, 1]$

$$\alpha(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega, \\ 0 & \text{otherwise,} \end{cases}$$

integrals over the physical geometry Ω can be expressed as integrals over Ω^{fict}

$$\int_{\Omega} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_{\Omega^{\text{fict}}} \alpha(\boldsymbol{x}) f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \tag{8.21}$$

for a suitable integrand $f : \Omega^{\text{fict}} \to \mathbb{R}$. The moment-fitting approach can be applied to define quadrature rules on Ω^{fict} to compute integrals over Ω. In particular, given the geometric mapping $\boldsymbol{F}$, such that $\Omega^{\text{fict}} = \boldsymbol{F}(\hat{\Omega}^{\text{fict}})$, integration rules can be obtained by solving the system

$$\begin{bmatrix} \hat{f}_1(\boldsymbol{\xi}_1) & \hat{f}_1(\boldsymbol{\xi}_2) & \dots & \hat{f}_1(\boldsymbol{\xi}_n) \\ \hat{f}_2(\boldsymbol{\xi}_1) & \hat{f}_2(\boldsymbol{\xi}_2) & \dots & \hat{f}_2(\boldsymbol{\xi}_n) \\ \vdots & \vdots & & \vdots \\ \hat{f}_m(\boldsymbol{\xi}_1) & \hat{f}_m(\boldsymbol{\xi}_2) & \dots & \hat{f}_m(\boldsymbol{\xi}_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_{\hat{\Omega}^{\text{fict}}} \hat{\alpha}(\boldsymbol{\xi}) \hat{f}_1(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \\ \int_{\hat{\Omega}^{\text{fict}}} \hat{\alpha}(\boldsymbol{\xi}) \hat{f}_2(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \\ \vdots \\ \int_{\hat{\Omega}^{\text{fict}}} \hat{\alpha}(\boldsymbol{\xi}) \hat{f}_m(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \end{bmatrix}, \tag{8.22}$$

for some integration points $\{\boldsymbol{\xi}_i\}_{i=1\dots n} \subset \hat{\Omega}^{\text{fict}}$ and weights $\{w_i\}_{i=1\dots n}$. Here, $\hat{\alpha}$ and $\hat{f}$ denote $\alpha \circ \boldsymbol{F}$, $f \circ \boldsymbol{F}$, respectively. Note that the integration points are distributed in the parametric fictitious domain.

The mass matrix entries are defined as

$$\begin{aligned} M_{ij} &= \int_{\Omega} \rho(\boldsymbol{x}) \; N_i(\boldsymbol{x}) \; N_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \\ &= \int_{\Omega^{\text{fict}}} \alpha(\boldsymbol{x}) \; \rho(\boldsymbol{x}) \; N_i(\boldsymbol{x}) \; N_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \\ &= \int_{\hat{\Omega}^{\text{fict}}} g_\alpha(\boldsymbol{\xi}) \; \hat{N}_i(\boldsymbol{\xi}) \; \hat{N}_j(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi}. \end{aligned} \tag{8.23}$$

The coefficient function $g_\alpha(\boldsymbol{\xi})$ now includes the domain indicator function $g_\alpha(\boldsymbol{\xi}) = \hat{\alpha}(\boldsymbol{\xi})\hat{\rho}(\boldsymbol{\xi}) \det(\boldsymbol{J})$, where $\boldsymbol{J}$ is the Jacobian of $\boldsymbol{F}$ and $\hat{\rho} = \rho \circ \boldsymbol{F}$.

Given the polynomial orders $\{p_d\}_{d=1\dots D}$ of the basis functions $\{{}^d\hat{N}_i\}_{d=1\dots D}$, the moment fitting equations in Equation (8.22) can be solved to accurately integrate the mass matrix

with $n = m = (2p_1 + 1)...(2p_D + 1)$ through the Lagrange polynomials, as explained in the previous section.

The resulting moment-fitting weights w_q can be used to integrate the mass matrix in the form given by Equation (8.9), where the diagonal matrix $\boldsymbol{G}$, $G_{qq} = \hat{\rho}(\boldsymbol{\xi}_q)\det(\boldsymbol{J})w_q$, does not include α, as this information is condensed in the moment-fitting weights w_q.

8.5 Stiffness matrix (heat conduction)

In the previous sections, the evaluation of the matrix-vector product $\boldsymbol{M}\boldsymbol{v}$ was exemplified for the mass matrix $\boldsymbol{M}$. In this section, the multiplication by heat-conduction stiffness matrices is discussed following Sangalli and Tani [2018]. Classical elastostatics can be treated similarly, as described in Antolin et al. [2015]; Hiemstra et al. [2019].

Given a conductivity law $\boldsymbol{\kappa}$, the Galerkin stiffness matrix $\boldsymbol{K} \in \mathbb{R}^{n_f \times n_f}$ for heat conduction can be written in the following form [Sangalli and Tani, 2018]

$$\begin{aligned} K_{ij} &= \int_{\Omega} \nabla N_i(\boldsymbol{x})^\top \boldsymbol{\kappa}(\boldsymbol{x}) \nabla N_j(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} \\ &= \int_{\hat{\Omega}} \nabla \hat{N}_i(\boldsymbol{\xi})^\top \hat{\boldsymbol{\kappa}}(\boldsymbol{\xi}) \nabla \hat{N}_j(\boldsymbol{\xi}) \,\mathrm{d}\boldsymbol{\xi} \\ &= \sum_{r=1}^{D}\sum_{s=1}^{D} \int_{\hat{\Omega}} \frac{\partial \hat{N}_i(\boldsymbol{\xi})}{\partial \xi_r} \, \hat{\kappa}_{rs}(\boldsymbol{\xi}) \, \frac{\partial \hat{N}_j(\boldsymbol{\xi})}{\partial \xi_s} \,\mathrm{d}\boldsymbol{\xi}, \end{aligned}$$

where $\hat{\boldsymbol{\kappa}}(\boldsymbol{\xi}) = \boldsymbol{J}^{-\top}\boldsymbol{\kappa}(\boldsymbol{x}(\boldsymbol{\xi}))\boldsymbol{J}^{-1}\,\det(\boldsymbol{J})$. The numerical quadrature over the integration points $\{\boldsymbol{\xi}_q\}_{q=1...n_q}$ and weights $\{w_q\}_{q=1...n_q}$ can be written as follows

$$K_{ij} \approx \tilde{K}_{ij} = \sum_{r=1}^{D}\sum_{s=1}^{D}\sum_{q=1}^{n_q} \frac{\partial \hat{N}_i(\boldsymbol{\xi}_q)}{\partial \xi_r} \, \hat{\kappa}_{rs}(\boldsymbol{\xi}_q) \, \frac{\partial \hat{N}_j(\boldsymbol{\xi}_q)}{\partial \xi_s} \, w_q,$$

resulting in the matrix form

$$\tilde{\boldsymbol{K}} = \sum_{r=1}^{D}\sum_{s=1}^{D} \boldsymbol{B}_{,r}^\top \hat{\boldsymbol{\kappa}}_{rs} \boldsymbol{B}_{,s}.$$

The matrices $\boldsymbol{B}_{,s} \in \mathbb{R}^{n_q \times n_f}$ and the diagonal matrix $\hat{\boldsymbol{\kappa}}_{rs} \in \mathbb{R}^{n_q \times n_q}$ are defined as

$$\begin{aligned} [\boldsymbol{B}_{,s}]_{qj} &= \frac{\partial \hat{N}_j}{\partial \xi_s}(\boldsymbol{\xi}_q), \\ [\hat{\boldsymbol{\kappa}}_{rs}]_{qq} &= \hat{\kappa}_{rs}(\boldsymbol{\xi}_q)\, w_q. \end{aligned}$$

Algorithm 8.4: Stiffness matrix-vector multiplication [Sangalli and Tani, 2018].

Input: ${}^{d}\boldsymbol{B}_{,r} \in \mathbb{R}^{{}^{d}n_q \times {}^{d}n_f},\ d, r = 1 \dots D$
$\hat{\boldsymbol{\kappa}}_{rs} \in \mathbb{R}^{n_q \times n_q},\ r, s = 1 \dots D$
$\boldsymbol{v} \in \mathbb{R}^{n_f}$
Output: $\boldsymbol{r} = \tilde{\boldsymbol{K}}\boldsymbol{v}$.

```
r = 0
for s=1,...,D do
    w_1 = (^D B_{,s} ⊗ ... ⊗ ^1 B_{,s}) v
    for r=1,...,D do
        w_2 = κ̂_{rs} w_1
        w_3 = (^D B_{,r}^T ⊗ ... ⊗ ^1 B_{,r}^T) w_2
        r = r + w_3
    end
end
```

If Assumption 1 and Assumption 2 hold, the tensor structure is transferred to $\boldsymbol{B}_{,r}$, yielding

$$\begin{aligned}\tilde{\boldsymbol{K}} &= \sum_{r=1}^{D}\sum_{s=1}^{D}\left({}^{D}\boldsymbol{B}_{,r}^{\top} \otimes \dots \otimes {}^{1}\boldsymbol{B}_{,r}^{\top}\right)\hat{\boldsymbol{\kappa}}_{rs}\left({}^{D}\boldsymbol{B}_{,s} \otimes \dots \otimes {}^{1}\boldsymbol{B}_{,s}\right) \\ &= \sum_{r=1}^{D}\left({}^{D}\boldsymbol{B}_{,r}^{\top} \otimes \dots \otimes {}^{1}\boldsymbol{B}_{,r}^{\top}\right)\sum_{s=1}^{D}\hat{\boldsymbol{\kappa}}_{rs}\left({}^{D}\boldsymbol{B}_{,s} \otimes \dots \otimes {}^{1}\boldsymbol{B}_{,s}\right),\end{aligned} \tag{8.24}$$

with

$${}^{d}\boldsymbol{B}_{,r} = \begin{cases} {}^{d}\boldsymbol{B}' & \text{if } r = d, \\ {}^{d}\boldsymbol{B} & \text{otherwise.} \end{cases}$$

Here, ${}^{d}\boldsymbol{B}$ is defined as in Equation (8.9), while ${}^{d}\boldsymbol{B}'$ is the matrix of derivatives ${}^{d}B'_{qj} = {}^{d}\hat{N}'_{j}({}^{d}\xi_q)$. According to Equation (8.24), the multiplication $\tilde{\boldsymbol{K}}\boldsymbol{v}$ can be computed as summarized in Algorithm 8.4. The cost for $\hat{n}_f$ basis functions and $\hat{n}_q$ integration points in each parametric direction becomes

$$(D + D^2)\left(\hat{n}_f\hat{n}_q^D + \hat{n}_f^2\hat{n}_q^{D-1} + \dots + \hat{n}_f^D\hat{n}_q\right) + D^2\hat{n}_q^D.$$

This strategy can be used in combination with local refinement after transforming the locally-refined basis to a reference tensor-product space. Moreover, it can be combined with immersed methods based on a tensor-product integration rule, as explained in Sections 8.3 and 8.4. Note that often the element mesh for immersed methods is a regular grid (see, e.g., Düster et al. [2017]; Parvizian et al. [2007]). In this case, the elements are axis-aligned regular cuboids, making the Jacobian $\boldsymbol{J}$ a diagonal matrix. If the conductivity $\boldsymbol{\kappa}$ is also diagonal (or isotropic), then $\hat{\kappa}_{rs}$ will be zero for $r \neq s$, and the two nested loops in Algorithm 8.4 can be reduced to one single loop.

8.5.1 Lagrange factorization

A further optimization introduced in Kronbichler and Kormann [2019] can be integrated into Algorithm 8.4. In particular, let ${}^d\boldsymbol{L}$ be such that

$$ {}^d\boldsymbol{B}' = {}^d\boldsymbol{L}\,{}^d\boldsymbol{B}. $$

This is possible as long as ${}^dn_q \geq {}^dn_f$ and ${}^d\boldsymbol{B}$ has full column rank. For polynomial bases, ${}^d\boldsymbol{L}$ can be generated from the Lagrange polynomials ${}^d\ell_j(\xi)$ (see Equation (8.18)) defined on the nodes given by the same integration points $\{{}^d\xi_q\}_{q=1\ldots{}^dn_q}$ used to evaluate ${}^dB_{qj} = {}^d\hat{N}_j({}^d\xi_q)$. In particular, ${}^d\boldsymbol{L}$ can be chosen as

$$ {}^dL_{iq} = {}^d\ell_i'({}^d\xi_q). $$

The multiplication by the stiffness matrix can be expanded as (see Equation (8.24))

$$ \tilde{\boldsymbol{K}}\boldsymbol{v} = \sum_{r=1}^{D} \left({}^D\boldsymbol{B}_{,r}^\top \otimes \ldots \otimes {}^1\boldsymbol{B}_{,r}^\top\right) \sum_{s=1}^{D} \hat{\boldsymbol{\kappa}}_{rs} \left({}^D\boldsymbol{B}_{,s} \otimes \ldots \otimes {}^1\boldsymbol{B}_{,s}\right) \boldsymbol{v} \tag{8.25} $$

$$ \begin{aligned} &= \left({}^D\boldsymbol{B}^\top \otimes \ldots \otimes {}^1\boldsymbol{B}^\top\right) \Bigg\{ \ldots \\ &\qquad \sum_{r=1}^{D} \left({}^D\boldsymbol{I}_{,r}^\top \otimes \ldots \otimes {}^1\boldsymbol{I}_{,r}^\top\right) \sum_{s=1}^{D} \hat{\boldsymbol{\kappa}}_{rs} \left({}^D\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^1\boldsymbol{I}_{,s}\right) \Bigg\} \left({}^D\boldsymbol{B} \otimes \ldots \otimes {}^1\boldsymbol{B}\right) \boldsymbol{v}, \end{aligned} \tag{8.26} $$

where

$$ {}^d\boldsymbol{I}_{,r} = \begin{cases} {}^d\boldsymbol{L} & \text{if } r = d, \\ \boldsymbol{I} & \text{otherwise.} \end{cases} $$

For example, for $D = 2$, isotropic conductivity, and diagonal Jacobian, Equation (8.26) becomes

$$ \begin{aligned} \tilde{\boldsymbol{K}} &= \left({}^2\boldsymbol{B}' \otimes {}^1\boldsymbol{B}\right)^\top \hat{\boldsymbol{\kappa}}_{11} \left({}^2\boldsymbol{B}' \otimes {}^1\boldsymbol{B}\right) + \left({}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B}'\right)^\top \hat{\boldsymbol{\kappa}}_{22} \left({}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B}'\right) \\ &= \left({}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B}\right)^\top \{ \quad \left({}^2\boldsymbol{L} \otimes \boldsymbol{I}\right)^\top \hat{\boldsymbol{\kappa}}_{11} \left({}^2\boldsymbol{L} \otimes \boldsymbol{I}\right) \ldots \\ &\qquad\qquad + \left(\boldsymbol{I} \otimes {}^1\boldsymbol{L}\right)^\top \hat{\boldsymbol{\kappa}}_{22} \left(\boldsymbol{I} \otimes {}^1\boldsymbol{L}\right) \quad \} \left({}^2\boldsymbol{B} \otimes {}^1\boldsymbol{B}\right). \end{aligned} $$

Note that in multiplication $\left({}^D\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^1\boldsymbol{I}_{,s}\right)\boldsymbol{v}$, only the matrix in the parametric direction s is different from the identity matrix. Therefore, if all parametric directions have the same number $\hat{n}_q$ of quadrature points, the cost of this operation is $\hat{n}_q^{D+1}$. The cost of Algorithms 8.4 and 8.5 is summarized in Table 8.1, assuming the same number of quadrature points $\hat{n}_q$ and basis functions $\hat{n}_f$ is used in each parametric direction. Under this assumption, it is worth using Equation (8.26) in three dimensions when

$$ \begin{aligned} 1 > \phi(\hat{n}_q, \hat{n}_f) &= \frac{\texttt{cost}\,[\text{Algorithm 8.5}]}{\texttt{cost}\,[\text{Algorithm 8.4}]} \\ &= \frac{2\hat{n}_f^3 + 2\hat{n}_f^2\hat{n}_q + 2\hat{n}_f\hat{n}_q^2 + 12\hat{n}_q^3 + 9\hat{n}_q^2}{12\hat{n}_f^3 + 12\hat{n}_f^2\hat{n}_q + 12\hat{n}_f\hat{n}_q^2 + 9\hat{n}_q^2}. \end{aligned} $$

Algorithm 8.5: Stiffness matrix-vector multiplication with Lagrange factorization.

Input: ${}^{d}\boldsymbol{B}_{,r} \in \mathbb{R}^{{}^{d}n_q \times {}^{d}n_f},\ d, r = 1 \dots D$
${}^{d}\boldsymbol{L} \in \mathbb{R}^{{}^{d}n_q \times {}^{d}n_q},\ d$
$\hat{\boldsymbol{\kappa}}_{rs} \in \mathbb{R}^{n_q \times n_q},\ r, s = 1 \dots D$
$\boldsymbol{v} \in \mathbb{R}^{n_f}$
Output: $\boldsymbol{r} = \tilde{\boldsymbol{K}}\boldsymbol{v}$.

$\boldsymbol{t} = \boldsymbol{0}$
$\boldsymbol{w}_1 = \left({}^{D}\boldsymbol{B} \otimes \dots \otimes {}^{1}\boldsymbol{B}\right)\boldsymbol{v}$ // using Algorithm 5.7
for s=1,...,D **do**
  $\boldsymbol{w}_2 = \left({}^{D}\boldsymbol{I}_{,s} \otimes \dots \otimes {}^{1}\boldsymbol{I}_{,s}\right)\boldsymbol{w}_1$ // using Algorithm 5.7 in one direction
  for r=1,...,D **do**
    $\boldsymbol{w}_3 = \hat{\boldsymbol{\kappa}}_{rs}\,\boldsymbol{w}_2$ // straightforward, as $\hat{\boldsymbol{\kappa}}_{rs}$ is diagonal
    $\boldsymbol{w}_4 = \left({}^{D}\boldsymbol{I}_{,r}^{\top} \otimes \dots \otimes {}^{1}\boldsymbol{I}_{,r}^{\top}\right)\boldsymbol{w}_3$ // using Algorithm 5.7 in one direction
    $\boldsymbol{t} = \boldsymbol{t} + \boldsymbol{w}_4$
  end
end
$\boldsymbol{r} = \left({}^{D}\boldsymbol{B}^{\top} \otimes \dots \otimes {}^{1}\boldsymbol{B}^{\top}\right)\boldsymbol{t}$ // using Algorithm 5.7

	no. multiplications
`cost`[Algorithm 8.4]	$(D + D^2)\left(\hat{n}_f\hat{n}_q^D + \hat{n}_f^2\hat{n}_q^{D-1} + \dots + \hat{n}_f^D\hat{n}_q\right) + D^2\hat{n}_q^D$
`cost`[Algorithm 8.5]	$2\left(\hat{n}_f\hat{n}_q^D + \hat{n}_f^2\hat{n}_q^{D-1} + \dots + \hat{n}_f^D\hat{n}_q\right) + (D + D^2)\hat{n}_q^{D+1} + D^2\hat{n}_q^D$

Table 8.1: Comparison between the number of floating-point multiplications for evaluating Algorithms 8.4 and 8.5 assuming the same number of quadrature points $\hat{n}_q$ and basis functions $\hat{n}_f$ is used in each parametric direction.

This condition holds when

$$\frac{\hat{n}_q}{\hat{n}_f} < \frac{1}{18}\left(5 + (5(646 + 27\sqrt{489}))^{\frac{1}{3}} + (5(646 - 27\sqrt{489}))^{\frac{1}{3}}\right) \approx 1.64671, \tag{8.27}$$

and for $D = 3, \hat{n}_q = \hat{n}_f = p + 1$, the number of multiplications will tend to be halved for increasing p

$$\phi = \frac{2(p+1)+1}{4(p+1)+1} \to \frac{1}{2}, \qquad p \to \infty. \tag{8.28}$$

Note that Equation (8.27) shows that this factorization is not advantageous for three-dimensional cut elements integrated with the linear moment fitting approach (see Section 8.4), where normally $\hat{n}_q = 2p+1$, $\hat{n}_f = p+1$, giving $\hat{n}_q/\hat{n}_f > 1.64671$ for $p > 1$. This approach is of advantage for untrimmed elements, where $\hat{n}_q = \hat{n}_f = p + 1$.

Further efficiency can be obtained for problems where the mass and stiffness matrices are summed together. In this case, the Lagrange factorization allows factoring out the

Algorithm 8.6: Stiffness matrix-vector multiplication with Lagrange factorization for diagonal conductivity and Jacobians. MATLAB's colon notation is used to indicate sub-matrices [MATLAB, 2019].

Input: ${}^{d}\boldsymbol{B}_{,r} \in \mathbb{R}^{{}^{d}n_q \times {}^{d}n_f},\ d, r = 1 \dots D$
${}^{d}\boldsymbol{L} \in \mathbb{R}^{{}^{d}n_q \times {}^{d}n_q},\ d$
$\hat{\boldsymbol{\kappa}}_{ss} \in \mathbb{R}^{n_q \times n_q},\ s = 1 \dots D$
$\boldsymbol{v} \in \mathbb{R}^{n_f}$
Output: $\boldsymbol{r} = \tilde{\boldsymbol{K}}\boldsymbol{v}$.

$\boldsymbol{t} = \boldsymbol{0}$
$\boldsymbol{w}_1 = \left({}^{D}\boldsymbol{B} \otimes \ldots \otimes {}^{1}\boldsymbol{B}\right)\boldsymbol{v}$
nleft $= {}^{2}n_q\, {}^{3}n_q \ldots {}^{D}n_q$
nright $= 1$
for $s = 1 \dots D$ **do**
  for $k = 0 \dots$ *nleft*-1 **do**
    for $j = 1 \dots$ *nright* **do**
      start $= k\, n_s$ nright $+ j$
      end $=$ start+nright $(n_i - 1)$
      ix $=$ (start : nright : end)
      $\boldsymbol{t}($ ix, : $) = \boldsymbol{t}($ ix, : $) + {}^{i}\boldsymbol{L}^{\top}\ \hat{\boldsymbol{\kappa}}_{ss}($ ix, ix $)\ {}^{i}\boldsymbol{L}\ \boldsymbol{v}($ ix, : $)$
    end
  end
  nleft $=$ nleft / $n_{\min\{i+1,D\}}$
  nright $=$ nright n_i
end
$\boldsymbol{r} = \left({}^{D}\boldsymbol{B}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{B}^{\top}\right)\boldsymbol{t}$

common terms $\left({}^{D}\boldsymbol{B} \otimes \ldots \otimes {}^{1}\boldsymbol{B}\right)$. Indeed, adding Equations (8.9) and (8.26) yields

$$\tilde{\boldsymbol{K}} + \tilde{\boldsymbol{M}} = \left({}^{D}\boldsymbol{B}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{B}^{\top}\right) \Bigg\{ \boldsymbol{G} + \ldots \sum_{r=1}^{D} \left({}^{D}\boldsymbol{I}_{,r}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,r}^{\top}\right) \sum_{s=1}^{D} \hat{\boldsymbol{\kappa}}_{rs} \left({}^{D}\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,s}\right) \Bigg\} \left({}^{D}\boldsymbol{B} \otimes \ldots \otimes {}^{1}\boldsymbol{B}\right).$$

Note also that in the case of diagonal conductivity law $\boldsymbol{\kappa}$ and diagonal Jacobians, the two loops in Algorithm 8.5 can be reduced to one single loop, as $\hat{\boldsymbol{\kappa}}_{rs} = 0$ for $r \neq s$. For example, see Algorithm 8.6, where the multiplications by $\left({}^{D}\boldsymbol{I}_{,s}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,s}^{\top}\right)$ and $\left({}^{D}\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,s}\right)$ are performed simultaneously for each parametric direction.

8.6 Enforcement of penalty boundary conditions

The application of displacement penalty boundary conditions adds to the system matrix a term of the form (see Equation (7.14))

$$T_{ij} = \int_\Gamma \beta \; N_i(\boldsymbol{x}) \; N_j(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \tag{8.29}$$

The costs can be reduced by combining the multiplication by the penalty term in Equation (8.29) and the multiplication by the stiffness matrix given in Equation (8.24). The surface integrals in Equation (8.29) are of the form

$$\int_\Gamma f \, \mathrm{d}\boldsymbol{x}, \tag{8.30}$$

where the integrand $f : \Omega \to \mathbb{R}$, $\Gamma \subset \Omega \subset \mathbb{R}^D$, is sufficiently smooth. Note that f is not defined only on Γ, but also on a volume Ω in $\mathbb{R}^D$ enclosing Γ. The moment fitting approach introduced in Section 8.4 can be adapted to integrate this kind of surface integrals with integration points distributed in the parameter space $\hat{\Omega}$ (with $\Omega = \boldsymbol{F}(\hat{\Omega})$ for some mapping $\boldsymbol{F}$) [Müller et al., 2013]. In particular, given the fixed locations $\{\boldsymbol{\xi}_i\}_{i=1\ldots n} \subset \hat{\Omega}$, $\boldsymbol{x}_i = \boldsymbol{F}(\boldsymbol{\xi}_i)$, the weights $\{w_i\}_{i=1\ldots n}$ are obtained by solving the system

$$\begin{bmatrix} f_1(\boldsymbol{x}_1) & f_1(\boldsymbol{x}_2) & \ldots & f_1(\boldsymbol{x}_n) \\ f_2(\boldsymbol{x}_1) & f_2(\boldsymbol{x}_2) & \ldots & f_2(\boldsymbol{x}_n) \\ \vdots & \vdots & & \vdots \\ f_m(\boldsymbol{x}_1) & f_m(\boldsymbol{x}_2) & \ldots & f_m(\boldsymbol{x}_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_\Gamma f_1 \, \mathrm{d}\Gamma \\ \int_\Gamma f_2 \, \mathrm{d}\Gamma \\ \vdots \\ \int_\Gamma f_m \, \mathrm{d}\Gamma \end{bmatrix}. \tag{8.31}$$

For polynomial integrands, the weights can be computed directly as integrals of the Lagrange polynomials, as explained in Section 8.4.

This approach factorizes together terms of the element system matrices and boundary conditions. Specifically, let $\{\boldsymbol{\xi}_i\}_{i=1\ldots n} \subset \hat{\Omega}$ be a fixed set of integration points forming a tensor product of the univariate integration points $\{{}^d\xi_q\}_{q=1\ldots {}^d n_q}$ in each parametric direction $d = 1\ldots D$. Moreover, let $\{w_i^v\}_{i=1\ldots n}$ be the weights obtained by solving Equation (8.17) with locations $\{\boldsymbol{\xi}_i\}_{i=1\ldots n}$ to compute the (volumetric) integrals in the element system matrix, and let $\{w_i^s\}_{i=1\ldots n}$ be the weights obtained by solving Equation (8.31) on the same locations $\{\boldsymbol{\xi}_i\}_{i=1\ldots n}$ to compute the (surface) integrals over Γ. Numerical integration of Equation (8.29) can be written in matrix form as

$$\boldsymbol{T} \approx \tilde{\boldsymbol{T}} = \left({}^D\boldsymbol{B}^\top \otimes \ldots \otimes {}^1\boldsymbol{B}^\top \right) \boldsymbol{G}^s \left({}^D\boldsymbol{B} \otimes \ldots \otimes {}^1\boldsymbol{B} \right), \tag{8.32}$$

where $\boldsymbol{G}^s$ is a diagonal matrix containing the surface integration weights, $G^s_{qq} = \beta w^s_q$, and ${}^dB_{qj} = {}^d\hat{N}_j({}^d\xi_q)$ is the usual collocated matrix, as defined in Equation (8.9). Equation (8.32) can be combined with Equation (8.26) to compute the element stiffness matrix

including the penalty term

$$\tilde{\boldsymbol{K}} + \tilde{\boldsymbol{T}} = \left({}^{D}\boldsymbol{B}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{B}^{\top}\right) \Bigg\{ \boldsymbol{G}^{s} + \ldots \\ \sum_{r=1}^{D} \left({}^{D}\boldsymbol{I}_{,r}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,r}^{\top}\right) \sum_{s=1}^{D} \hat{\boldsymbol{\kappa}}^{\boldsymbol{v}}{}_{rs} \left({}^{D}\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,s}\right) \Bigg\} \left({}^{D}\boldsymbol{B} \otimes \ldots \otimes {}^{1}\boldsymbol{B}\right), \tag{8.33}$$

where $\hat{\boldsymbol{\kappa}}^{v}_{rs}$ contains the volumetric weights $\{w_q^v\}_{i=q\ldots n}$

$$\left[\hat{\boldsymbol{\kappa}}^{\boldsymbol{v}}{}_{rs}\right]_{qq} = \hat{\kappa}_{rs}(\boldsymbol{\xi}_q)\, w_q^v. \tag{8.34}$$

Note that using a quadrature rule defined in the volume Ω to compute the surface integrals in Equation (8.31) allows writing $\tilde{\boldsymbol{K}}$ and $\tilde{\boldsymbol{T}}$ in terms of the same matrices ${}^{d}\boldsymbol{B}$, that can be factored out in Equation (8.33). For problems involving mass and stiffness matrices, the evaluated shape function ${}^{d}\boldsymbol{B}$ can be factored out of the sum $\tilde{\boldsymbol{K}} + \tilde{\boldsymbol{M}} + \tilde{\boldsymbol{T}}$.

This approach can be used with hierarchically refined bases using the extraction operators $\boldsymbol{C}$ and the tensor-product reference basis matrices $\boldsymbol{S} = {}^{D}\boldsymbol{S} \otimes \ldots \otimes {}^{1}\boldsymbol{S}$ (see Equations (8.8), (8.12), and (8.14)). In this case, $\tilde{\boldsymbol{K}} + \tilde{\boldsymbol{T}}$ takes the form

$$\tilde{\boldsymbol{K}} + \tilde{\boldsymbol{T}} = \boldsymbol{C} \left({}^{D}\boldsymbol{S}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{S}^{\top}\right) \Bigg\{ \boldsymbol{G}^{s} + \ldots \\ \sum_{r=1}^{D} \left({}^{D}\boldsymbol{I}_{,r}^{\top} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,r}^{\top}\right) \sum_{s=1}^{D} \hat{\boldsymbol{\kappa}}^{v}_{rs} \left({}^{D}\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^{1}\boldsymbol{I}_{,s}\right) \Bigg\} \left({}^{D}\boldsymbol{S} \otimes \ldots \otimes {}^{1}\boldsymbol{S}\right) \boldsymbol{C}^{\top}. \tag{8.35}$$

Equation (8.35) can be evaluated as in Algorithm 8.7.

Algorithm 8.7: System matrix (stiffness and penalty term) matrix-vector multiplication.

$\boldsymbol{w}_1 = \boldsymbol{C}^\top \boldsymbol{v}$ // using Algorithm 5.13
$\boldsymbol{w}_2 = \left({}^D\boldsymbol{S} \otimes \ldots \otimes {}^1\boldsymbol{S}\right) \boldsymbol{w}_1$ // using Algorithm 5.7 or 5.8
$\boldsymbol{w}_3 = \boldsymbol{G}^s \boldsymbol{w}_2$ // straightforward, as $\boldsymbol{G}^s$ is diagonal
for s=1,...,D **do**
  $\boldsymbol{w}_4 = \left({}^D\boldsymbol{I}_{,s} \otimes \ldots \otimes {}^1\boldsymbol{I}_{,s}\right) \boldsymbol{w}_2$ // using Algorithm 5.7 in one direction
  for r=1,...,D **do**
    $\boldsymbol{w}_5 = \hat{\boldsymbol{\kappa}}_{rs} \boldsymbol{w}_4$ // straightforward, as $\hat{\boldsymbol{\kappa}}_{rs}$ is diagonal
    $\boldsymbol{w}_6 = \left({}^D\boldsymbol{I}_{,r}^\top \otimes \ldots \otimes {}^1\boldsymbol{I}_{,r}^\top\right) \boldsymbol{w}_5$ // using Algorithm 5.7 in one direction
    $\boldsymbol{w}_3 = \boldsymbol{w}_3 + \boldsymbol{w}_6$
  end
end
$\boldsymbol{w}_7 = \left({}^D\boldsymbol{S}^\top \otimes \ldots \otimes {}^1\boldsymbol{S}^\top\right) \boldsymbol{w}_3$ // using Algorithm 5.7 or 5.8
$\boldsymbol{r} = \boldsymbol{C}\boldsymbol{w}_7$ // using Algorithm 5.10

Chapter 9

Conclusion

9.1 Summary of the research

This thesis aims to develop several technologies supporting the computational framework composed of the finite cell method, isogeometric analysis, and hierarchical local refinement. In particular, this work provides the contributions discussed in the following subsections.

9.1.1 The multi-level Bézier extraction

It is presented a formulation that brings multi-level local refinement closer to traditional finite element implementations.

In particular, the hierarchical basis functions are viewed as a linear combination of a set of standard functions common to each element. These reference functions also define a direct isoparametric geometric mapping from a reference parameter space to the element domain. The only modification required in a standard finite element implementation based on a common reference space is a linear transformation.

The multi-level Bézier extraction exhibits the same properties as the original Bézier extraction. Therefore, this approach facilitates integrating hierarchical isogeometric local refinement into existing classical implementations by offering the same advantages as the original Bézier extraction. However, the software must be able to handle element matrices of different sizes.

The approach is presented focusing on its application to (truncated) hierarchical B-splines and NURBS. However, the method is valid for any sequence of nested spaces, and its application to degree elevation is outlined.

9.1.2 Algorithms for the multi-level Bézier extraction

It is formulated an approach for (truncated) hierarchical B-splines in which

- the univariate extraction-operators are computed once for each parametric direction and hierarchical level.

- The following operations are formulated using only the univariate operators, without explicitly constructing and storing the full tensor-product operator:
  - extraction of functions,
  - degree-of-freedom projection for refinement, and
  - degree-of-freedom projection for coarsening (modified Bézier projection).

These operations are formulated by iterative algorithms through the refinement hierarchy, exploiting the level tensor-product structure.

Algorithms for constructing the univariate multi-level extraction operators are formulated based on standard knot-insertion techniques such as the Boehm and Oslo methods. Their algorithmic complexity is studied and compared.

In the case of mesh coarsening, a leveraging of the tensor-product structure is achieved by modifying the Bézier projection, where the product of rectangular operators is substituted by the product of square Kronecker operators. The algorithmic complexity of the proposed algorithms is studied, and the advantage over the explicit computation of the extraction operator is estimated.

The proposed approach is compared with an explicit computation of the extraction operator in both a linear transient and a nonlinear example. For cubic shape functions, the extraction time is reduced by a factor of 13 and 10, respectively. The quartic basis functions yield a reduction factor equal to 31.

9.1.3 Hierarchical local refinement of trimmed isogeometric finite cell analyses

Local refinement is classically used to efficiently resolve small-scale features. This work additionally shows that the considered hierarchical refinement can be used to mitigate FCM-specific difficulties such as

- the unphysical coupling between the sides of a thin hole and
- the overconstraining induced by weak boundary conditions.

The local refinement process defined by (truncated) hierarchical B-splines and NURBS replaces basis functions having large support with functions having smaller support. This effect allows removing the unphysical coupling of the numerical solution between two disconnected sides of thin holes, which can occur in the analysis of trimmed geometries.

Moreover, a numerical example shows how local refinement enriches the solution space in small regions following trimming curves immersed in the computational mesh. This procedure can mitigate the over-constraining effects caused by trimming curves subject to penalty boundary conditions.

The use of local refinement is demonstrated through various shell examples where the trimmed geometry includes sharp features, kinks, and small-scale details. The geometries are defined by files exported in the standard STEP format [ISO 10303-11:1994, 1994].

These examples demonstrate that the proposed strategy for local refinement can favor the direct analysis of B-Rep models, streamlining the design-through-analysis loop.

9.1.4 Reactions on trimmed locally-refined meshes

Locally-refined isogeometric finite cell analysis presents several differences compared to the standard finite element method. In the context of reaction forces, the relevant differences include:

- the isogeometric functions are not based on a "nodal" definition typical of the classic Lagrange finite elements,
- the boundary is generally immersed in the computational mesh and is subject to weak boundary conditions,
- the hierarchical B-splines do not form a partition of unity.

These differences make the classic methods for calculating the "nodal forces" inapplicable.

This thesis shows how the standard algorithm to compute the reactions can be interpreted as testing a variational form with specific test functions. This view serves as a basis for calculating the conservative reactions from solutions described by trimmed bases that do not form a partition of unity and are not defined by "nodes". It is shown that the equilibrium information can be extracted from the finite element solution to obtain a total reaction that balances the known forces. The proposed strategy is formulated in a general framework and then applied to the finite cell method in combination with (truncated) hierarchical B-splines and integrated Legendre (p-FEM) basis functions.

The approach is shown to be superconvergent and approximates with higher accuracy the total flux in a smooth two-dimensional problem for both penalty and Nitsche boundary conditions.

The applicability of the method is demonstrated on a trimmed linear Kirchhoff-Love shell example and a three-dimensional linear heat-conduction problem, where the geometry is defined by a detailed STL file with smooth wavy boundaries and several internal cavities.

9.1.5 Matrix-free approach to locally-refined finite cell analyses

The multi-level Bézier extraction algorithms are finally combined to formulate procedures for multiplying a vector by the finite element system matrix. These procedures are used to develop a framework suitable for matrix-free iterative solvers combining

- local refinement,
- trimming through the finite cell method, and
- sum factorization.

The sum factorization is enabled by using a tensor-product basis functions common to all elements and using tensor-product quadrature rules for the finite cell method. The first ingredient is offered naturally by the developed multi-level Bézier extraction.

The procedure is explained for the mass and stiffness matrices. A Lagrange factorization reduces the cost of penalty boundary conditions.

Finally, the complexity of the proposed algorithms is studied in terms of the number of floating-point multiplications.

9.2 Future research

The methods developed in this thesis were demonstrated through simple examples aiming at providing a proof of concept. Further possible research directions include:

- The discussed local refinement is valid for any sequence of nested spaces. These include discretizations composed of refined functions with lower continuity and of higher order. A simple benchmark with varying continuity across the refinement levels can be found in Kollmannsberger et al. [2020]. A similar study including multiple polynomial orders could be performed on selected benchmarks including singularities and stress concentrations.
- The multi-level Bézier extraction operator can enable pre-integration (see Yang et al. [2012a,b]) for local refinement. Indeed, the developed approach attributes to each element a common set of basis functions that can be pre-integrated. The locally-refined element matrices can be obtained by extraction of the pre-integrated matrices.
- The presented matrix-free framework focused only on the matrix-vector multiplication needed by iterative solvers. However, appropriate preconditioners are necessary to reduce the number of iterations needed for convergence. One possible research direction is to formulate matrix-free algorithms for the Schwarz-type preconditioners developed in de Prenter et al. [2019a,b]; Jomo et al. [2019]. These preconditioners have shown to be effective for large high-order finite cell analyses.
- The matrix-free algorithms should be tested on parallel distributed-memory systems for large-scale examples. This investigation would further assess the benefit of the approach for practical examples.

Appendix A

Algorithms for the knot insertion refinement operators

Algorithm A.1: Element Knot Insertion Operators (using Boehm's method).

Input: p: spline degree
$\boldsymbol{\Xi}^{(1)} = (\xi_1^{(1)}, \ldots, \xi_{m^{(1)}}^{(1)})$: coarse knot vector
$\boldsymbol{\Xi} = (\xi_1, \ldots, \xi_m)$: new knots to be inserted

Output: n_e: number of non-empty knot spans
$\boldsymbol{R}^e = \boldsymbol{R}^{e,(1,2)}$, $e = 1, \ldots, n_e$: local element operators
$\boldsymbol{\Xi}^{(2)} = (\xi_1^{(2)}, \ldots, \xi_{m^{(2)}}^{(2)})$: fine knot vector

$i = p + 1$; $k = p + 1$; $j = 0$; $e = 1$
for $k = 1 \ldots p + 1$ **do**
  $\xi_k^{(2)} = \xi_k^{(1)}$
end
$\boldsymbol{R}^1 = \boldsymbol{I}$;
while $i < m^{(1)}$ **or** $j < m$
  $nmv(e) = 0$; $om = 1$
  while $i + 1 + om \leq m^{(1)}$ **and** $\xi_{i+1}^{(1)} == \xi_{i+1+om}^{(1)}$ **do**
    $om = om + 1$
  end
  while $j < m$ **and** $\xi_{j+1} \leq \xi_{i+1}^{(1)}$
    $nm = 0$
    while j+1+nm≤m **and** $\xi_{j+1} == \xi_{j+1+nm}$ **do**
      $nm = nm + 1$
    end
    $nmv(e) = nm$
    if $\xi_{j+1} == \xi_{i+1}^{(1)}$ **then**
      $omv(e) = om$
    else
      $omv(e) = 0$
    end

```
1
2
3       for r = 0, ..., nm-1 do
4           R^{e+1} (p+1-omv(e),p+1-nm-omv(e)+r+1)=1
5           for l=0, ..., -r+1 do
6               α = (ξ^(1)_{i+p+l} - ξ_{j+1})/(ξ^(1)_{i+p+l} - ξ^(2)_{k+l})
7               ind2=p+1+l+r-nm-omv(e)
8               R^{e+1}(:,ind2)= α R^{e+1}(:,ind2-1) + (1-α) R^{e+1}(:,ind2)
9           end
10          for l=-r,...,-p+1 do
11              α = (ξ^(1)_{i+p+l} - ξ_{j+1})/(ξ^(1)_{i+p+l} - ξ^(2)_{k+l});
12              ind2=p+1+l+r;
13              R^e(:,ind2)= α R^e(:,ind2-1) + (1-α) R^e(:,ind2);
14          end
15          R^{e+1}(1:p+1-omv(e),1:p+1-nm-omv(e))=
16                R^e(1+omv(e):p+1,omv(e)+nm+1:p+1);
17          ξ^(2)_{k+1} = ξ_{j+1};
18          k=k+1;
19          j=j+1;
20      end
21      if omv(e) == 0 then
22          e = e+1;
23          nmv(e) = 0;
24      end
25  end
26  if i < m^(1) then
27      for t=0 ... om-1 do
28          ξ^(2)_{k+1} = ξ^(1)_{i+1}
29          R^{e+1}(p+1-t,p+1-t)= 1;
30          k = k + 1; i = i + 1;
31      end
32      omv(e)=om;
33      R^{e+1}(1:p+1-omv(e),1:p+1-nmv(e)-omv(e))=
34            R^e(1+omv(e):p+1,omv(e)+nmv(e)+1:p+1)
35      e = e+1
36  end
37 end
38 for f = e-1,...,2 do
39  R^{f-1}(1+omv(f-1):p+1,2+nmv(f-1)+omv(f-1):p+1)=
40        R^f(1:p+1-omv(f-1),2:p+1-nmv(f-1)-omv(f-1))
41 end
42 n_e = e − 1
```

Appendix B

Parametric definition of the elastoplastic perforated plate

The geometry of the numerical example given in Section 5.7.2 consists of a NURBS patch defined by the open-knot vectors

$$^{x}\boldsymbol{\Xi} = (-1,-1,-1,0,0,1,1,1), \tag{B.1}$$
$$^{y}\boldsymbol{\Xi} = (-1,-1,1,1), \tag{B.2}$$
$$^{z}\boldsymbol{\Xi} = {}^{y}\boldsymbol{\Xi}, \tag{B.3}$$

and control points

$$\boldsymbol{P} = \begin{bmatrix}
40 & 0 & 0 & 1 \\
40 & 60(\sqrt{2}-1) & 0 & (1+1/\sqrt{2})/2 \\
100-60/\sqrt{2} & 60/\sqrt{2} & 0 & 1 \\
100-60(\sqrt{2}-1) & 60 & 0 & (1+1/\sqrt{2})/2 \\
100 & 60 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 100 & 0 & 1 \\
0 & 200 & 0 & 1 \\
50 & 200 & 0 & 1 \\
100 & 200 & 0 & 1 \\
40 & 0 & 10 & 1 \\
40 & 60(\sqrt{2}-1) & 10 & (1+1/\sqrt{2})/2 \\
100-60/\sqrt{2} & 60/\sqrt{2} & 10 & 1 \\
100-60(\sqrt{2}-1) & 60 & 10 & (1+1/\sqrt{2})/2 \\
100 & 60 & 10 & 1 \\
0 & 0 & 10 & 1 \\
0 & 100 & 10 & 1 \\
0 & 200 & 10 & 1 \\
50 & 200 & 10 & 1 \\
100 & 200 & 10 & 1
\end{bmatrix}. \tag{B.4}$$

Bibliography

Abedian, A., Parvizian, J., Düster, A., Khademyzadeh, H., and Rank, E. (2013a). Performance of Different Integration Schemes in Facing Discontinuities in the Finite Cell Method. *International Journal of Computational Methods*, 10(03):1350002.

Abedian, A., Parvizian, J., Düster, A., and Rank, E. (2013b). The finite cell method for the J2 flow theory of plasticity. *Finite Elements in Analysis and Design*, 69:37–47.

Abedian, A., Parvizian, J., Düster, A., and Rank, E. (2014). Finite Cell Method Compared to H-Version Finite Element Method for Elasto-Plastic Problems. *Applied Mathematics and Mechanics*, 35(10):1239–1248.

Akira, M. (1986). A mixed finite element method for boundary flux computation. *Computer Methods in Applied Mechanics and Engineering*, 57(2):239–243.

Antolin, P., Buffa, A., Calabrò, F., Martinelli, M., and Sangalli, G. (2015). Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization. *Computer Methods in Applied Mechanics and Engineering*, 285:817 – 828.

Antolin, P., Buffa, A., Puppi, R., and Wei, X. (2019). Overlapping multi-patch isogeometric method with minimal stabilization.

Apostolatos, A., Breitenberger, M., Wüchner, R., and Bletzinger, K.-U. (2015). Domain decomposition methods and Kirchhoff-Love shell multipatch coupling in isogeometric analysis. In Jüttler, B. and Simeon, B., editors, *Isogeometric Analysis and Applications 2014*, pages 73–101. Springer International Publishing.

Apostolatos, A., Schmidt, R., Wüchner, R., and Bletzinger, K.-U. (2014). A Nitsche-type formulation and comparison of the most common domain decomposition methods in isogeometric analysis. *International Journal for Numerical Methods in Engineering*, 97(7):473–504.

Apprich, C., Höllig, K., Hörner, J., Keller, A., and Yazdani, E. N. (2014). Finite Element Approximation with Hierarchical B-Splines. In *Curves and Surfaces*, pages 1–15. Springer, Cham.

Babuška, I. (1973). The Finite Element Method with Penalty. *Mathematics of Computation*, 27(122):221–228.

Babuška, I. and Miller, A. (1984). The post-processing approach in the finite element methodpart 1: Calculation of displacements, stresses and other higher derivatives of the displacements. *International Journal for Numerical Methods in Engineering*, 20(6):1085–1109.

Barrett, J. W. and Elliott, C. M. (1987). Total flux estimates for a finite-element approximation of elliptic equations. *IMA journal of numerical analysis*, 7(2):129–148.

Bathe, K. J. (2007). *Finite Element Procedures*. Prentice Hall, New Jersey.

Bazilevs, Y., Calo, V. M., Cottrell, J. A., Evans, J. A., Hughes, T. J. R., Lipton, S., Scott, M. A., and Sederberg, T. W. (2010). Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5–8):229–263.

Bazilevs, Y., Hsu, M.-C., and Scott, M. (2012). Isogeometric fluid–structure interaction analysis with emphasis on non-matching discretizations, and with application to wind turbines. *Computer Methods in Applied Mechanics and Engineering*, 249-252:28–41. Higher Order Finite Element and Isogeometric Methods.

Bazilevs, Y. and Hughes, T. J. R. (2007). Weak imposition of Dirichlet boundary conditions in fluid mechanics. *Computers & Fluids*, 36(1):12–26.

Beirão da Veiga, L., Buffa, A., Cho, D., and Sangalli, G. (2012). Analysis-suitable t-splines are dual-compatible. *Computer Methods in Applied Mechanics and Engineering*, 249-252:42–51. Higher Order Finite Element and Isogeometric Methods.

Beirão da Veiga, L., Buffa, A., Sangalli, G., and Vázquez Hernández, R. (2013). Analysis-suitable t-splines of arbitrary degree: definition, linear independence and approximation properties. *Mathematical Models and Methods in Applied Sciences*, 23(11):1979–2003.

Boehm, W. (1985). On the efficiency of knot insertion algorithms. *Computer Aided Geometric Design*, 2(1–3):141–143.

Bog, T., Zander, N., Kollmannsberger, S., and Rank, E. (2015). Normal Contact with High Order Finite Elements and a Fictitious Contact Material. *Computers & Mathematics with Applications*, 70(7):1370–1390.

Bog, T., Zander, N., Kollmannsberger, S., and Rank, E. (2018). Weak imposition of frictionless contact constraints on automatically recovered high-order, embedded interfaces using the finite cell method. *Computational Mechanics*, 61(4):385–407.

Borden, M. J., Scott, M. A., Evans, J. A., and Hughes, T. J. R. (2011). Isogeometric finite element data structures based on Bézier extraction of NURBS. *International Journal for Numerical Methods in Engineering*, 87(1-5):15–47.

Bornemann, P. and Cirak, F. (2013). A subdivision-based implementation of the hierarchical b-spline finite element method. *Computer Methods in Applied Mechanics and Engineering*, 253:584–598.

Bracco, C., Buffa, A., Giannelli, C., and Vázquez, R. (2019). Adaptive isogeometric methods with hierarchical splines: An overview. *Discrete & Continuous Dynamical Systems-A*, 39(1):241.

Breitenberger, M., Apostolatos, A., Philipp, B., Wüchner, R., and Bletzinger, K. U. (2015). Analysis in computer aided design: Nonlinear isogeometric B-Rep analysis of shell structures. *Computer Methods in Applied Mechanics and Engineering*, 284:401–457.

Bressan, A. and Mokriš, D. (2017). A versatile strategy for the implementation of adaptive splines. In *Mathematical Methods for Curves and Surfaces*, pages 42–73. Springer International Publishing.

Brezzi, F., Hughes, T. J. R., and Süli, E. (2001). Variational approximation of flux in conforming finite element methods for elliptic partial differential equations : a model problem. *Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti Lincei. Matematica e Applicazioni*, 12(3):159–166.

Buffa, A. and Giannelli, C. (2016). Adaptive isogeometric methods with hierarchical splines: Error estimator and convergence. *Mathematical Models and Methods in Applied Sciences*, 26(01):1–25.

Cai, S., Zhang, W., Zhu, J., and Gao, T. (2014). Stress constrained shape and topology optimization with fixed mesh: A B-spline finite cell method combined with level set function. *Computer Methods in Applied Mechanics and Engineering*, 278:361–387.

Calabrò, F., Loli, G., Sangalli, G., and Tani, M. (2019). *Quadrature Rules in the Isogeometric Galerkin Method: State of the Art and an Introduction to Weighted Quadrature*, pages 43–55. Springer International Publishing, Cham.

Calabrò, F., Sangalli, G., and Tani, M. (2017). Fast formation of isogeometric galerkin matrices by weighted quadrature. *Computer Methods in Applied Mechanics and Engineering*, 316:606 – 622. Special Issue on Isogeometric Analysis: Progress and Challenges.

Carey, G. (1982). Derivative calculation from finite element solutions. *Computer Methods in Applied Mechanics and Engineering*, 35(1):1–14.

Carey, G., Chow, S., and Seager, M. (1985). Approximate boundary-flux calculations. *Computer Methods in Applied Mechanics and Engineering*, 50(2):107–120.

Carraturo, M., Giannelli, C., Reali, A., and Vázquez, R. (2019). Suitably graded thb-spline refinement and coarsening: Towards an adaptive isogeometric analysis of additive manufacturing processes. *Computer Methods in Applied Mechanics and Engineering*, 348:660 – 679.

Carraturo, M., Jomo, J., Kollmannsberger, S., Reali, A., Auricchio, F., and Rank, E. (2020). Modeling and experimental validation of an immersed thermo-mechanical part-scale analysis for laser powder bed fusion processes. *Additive Manufacturing*, 36:101498.

Casciola, G. and Romani, L. (2007). A general matrix representation for non-uniform B-spline subdivision with boundary control. Techincal report, ALMA-DL University of Bologna. amsacta.unibo.it/2532/.

Casquero, H., Liu, L., Zhang, Y., Reali, A., Kiendl, J., and Gómez, H. (2017). Arbitrary-degree T-splines for isogeometric analysis of fully nonlinear Kirchhoff-Love shells. *Computer-Aided Design*, 82:140 – 153. Isogeometric Design and Analysis.

Cirak, F. (2006). Subdivision shells. In Motasoares, C. A., Martins, J. A. C., Rodrigues, H. C., Ambrósio, J. A. C., Pina, C. A. B., Motasoares, C. M., Pereira, E. B. R., and Folgado, J., editors, *III European Conference on Computational Mechanics*, pages 395–395, Dordrecht. Springer Netherlands.

Coradello, L. (2016). *Implementation of a high-order Kirchhoff-Love shell: a comparison of iga and p-fem.* Master's thesis, Technische Universität München.

Coradello, L., Antolin, P., Vázquez, R., and Buffa, A. (2020a). Adaptive isogeometric analysis on two-dimensional trimmed domains based on a hierarchical approach. *Computer Methods in Applied Mechanics and Engineering*, 364:112925.

Coradello, L., D'Angella, D., Carraturo, M., Kiendl, J., Kollmannsberger, S., Rank, E., and Reali, A. (2020b). Hierarchically refined isogeometric analysis of trimmed shells. *Computational Mechanics*, 66:431–447.

Cottrell, J. A., Hughes, T. J. R., and Bazilevs, Y. (2009). *Isogeometric Analysis: Toward Integration of CAD and FEA.* John Wiley & Sons.

D'Angella, D., Kollmannsberger, S., Rank, E., and Reali, A. (2018). Multi-level Bézier extraction for hierarchical local refinement of Isogeometric Analysis. *Computer Methods in Applied Mechanics and Engineering*, 328:147–174.

D'Angella, D. and Reali, A. (2020). Efficient extraction of hierarchical b-splines for local refinement and coarsening of isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 367:113131.

Dauge, M., Düster, A., and Rank, E. (2015). Theoretical and Numerical Investigation of the Finite Cell Method. *J. Sci. Comput.*, 65(3):1039–1064.

De Borst, R., Crisfield, M. A., Remmers, J. J., and Verhoosel, C. V. (2012). *Nonlinear finite element analysis of solids and structures.* John Wiley & Sons, second edition.

de Prenter, F., Lehrenfeld, C., and Massing, A. (2018). A note on the stability parameter in Nitsche,s method for unfitted boundary value problems. *Computers & Mathematics with Applications*, 75(12):4322 – 4336.

de Prenter, F., Verhoosel, C., and van Brummelen, E. (2019a). Preconditioning immersed isogeometric finite element methods with application to flow problems. *Computer Methods in Applied Mechanics and Engineering*, 348:604631.

de Prenter, F., Verhoosel, C., van Zwieten, G., and van Brummelen, E. (2017). Condition number analysis and preconditioning of the finite cell method. *Computer Methods in Applied Mechanics and Engineering*, 316:297–327. Special Issue on Isogeometric Analysis: Progress and Challenges.

de Prenter, F., Verhoosel, C. V., van Brummelen, E. H., Evans, J. A., Messe, C., Benzaken, J., and Maute, K. (2019b). Multigrid solvers for immersed finite element methods and immersed isogeometric analysis. *Computational Mechanics*, 65(3):807838.

de Souza Neto, E., Perić, P., and Owen, P. (2009). *Computational Methods for Plasticity: Theory and Applications*. John Wiley & Sons.

Di Stolfo, P., Schröder, A., Zander, N., and Kollmannsberger, S. (2016). An easy treatment of hanging nodes in hp-finite elements. *Finite Elements in Analysis and Design*, 121:101–117.

Doha, E., Bhrawy, A., and Saker, M. (2011). Integrals of bernstein polynomials: An application for the solution of high even-order differential equations. *Applied Mathematics Letters*, 24(4):559 – 565.

Dokken, T., Lyche, T., and Pettersen, K. F. (2013). Polynomial splines over locally refined box-partitions. *Computer Aided Geometric Design*, 30(3):331 – 356.

Düster, A., Parvizian, J., Yang, Z., and Rank, E. (2008). The finite cell method for three-dimensional problems of solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 197(45–48):3768–3782.

Düster, A., Rank, E., and Szabó, B. (2017). *The p-Version of the Finite Element and Finite Cell Methods*, pages 1–35. John Wiley & Sons.

Elhaddad, M., Zander, N., Bog, T., Kudela, L., Kollmannsberger, S., Kirschke, J., Baum, T., Ruess, M., and Rank, E. (2018). Multi-level hp-finite cell method for embedded interface problems with application in biomechanics. *International Journal for Numerical Methods in Biomedical Engineering*, 34(4):e2951.

Elhaddad, M., Zander, N., Kollmannsberger, S., Shadavakhsh, A., Nübel, V., and Rank, E. (2015). Finite Cell Method: High-Order Structural Dynamics for Complex Geometries. *International Journal of Structural Stability and Dynamics*, 15(07):1540018.

Embar, A., Dolbow, J., and Harari, I. (2010). Imposing Dirichlet boundary conditions with Nitsche's method and spline-based finite elements. *International Journal for Numerical Methods in Engineering*, 83(7):877–898.

Evans, E. J., Scott, M. A., Li, X., and Thomas, D. C. (2015). Hierarchical T-splines: Analysis-suitability, Bézier extraction, and application as an adaptive basis for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:1–20.

Fachinotti, V. D., Anca, A. A., and Cardona, A. (2011). Analytical solutions of the thermal field induced by moving double-ellipsoidal and double-elliptical heat sources in a semi-infinite body. *International Journal for Numerical Methods in Biomedical Engineering*, 27(4):595–607.

Farouki, R. T. and Neff, C. A. (1990). On the numerical condition of bernstein-bézier subdivision processes. *Mathematics of Computation*, 55(192):637–647.

Fernandes, P., Plateau, B., and Stewart, W. J. (1998). Efficient descriptor-vector multiplications in stochastic automata networks. *J. ACM*, 45(3):381–414.

Forsey, D. R. and Bartels, R. H. (1988). Hierarchical B-spline Refinement. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pages 205–212, New York, NY, USA. ACM.

Garau, E. M. and Vázquez, R. (2016). Algorithms for the implementation of adaptive isogeometric methods using hierarchical splines. Techincal Report 16-08, IMATI-CNR, Pavia.

Garau, E. M. and Vázquez, R. (2018). Algorithms for the implementation of adaptive isogeometric methods using hierarchical b-splines. *Applied Numerical Mathematics*, 123:58 – 87.

Garhuom, W., Hubrich, S., Radtke, L., and Düster, A. (2020). A remeshing strategy for large deformations in the finite cell method. *Computers & Mathematics with Applications*, 80(11):2379–2398. High-Order Finite Element and Isogeometric Methods 2019.

Giannelli, C., Jüttler, B., Kleiss, S. K., Mantzaflaris, A., Simeon, B., and Špeh, J. (2016). THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 299:337–365.

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

Giannelli, C., Jüttler, B., and Speleers, H. (2013). Strongly stable bases for adaptively refined multilevel spline spaces. *Advances in Computational Mathematics*, 40(2):459–490.

Greiner, G. and Hormann, K. (1997). Interpolating and Approximating Scattered 3D-data with Hierarchical Tensor Product B-Splines. In *In Surface Fitting and Multiresolution Methods*, pages 163–172. Vanderbilt University Press.

Gresho, P. M., Lee, R. L., Sani, R. L., Maslanik, M. K., and Eaton, B. E. (1987). The consistent galerkin fem for computing derived boundary quantities in thermal and or fluids problems. *International Journal for Numerical Methods in Fluids*, 7(4):371–394.

Griebel, M. and Schweitzer, M. A. (2003). *A Particle-Partition of Unity Method Part V: Boundary Conditions*, pages 519–542. Springer Berlin Heidelberg, Berlin, Heidelberg.

Guo, Y., Heller, J., Hughes, T. J. R., Ruess, M., and Schillinger, D. (2018). Variationally consistent isogeometric analysis of trimmed thin shells at finite deformations, based on the step exchange format. *Computer Methods in Applied Mechanics and Engineering*, 336:39 – 79.

Guo, Y. and Ruess, M. (2015). Weak dirichlet boundary conditions for trimmed thin isogeometric shells. *Computers & Mathematics with Applications*, 70(7):1425–1440. High-Order Finite Element and Isogeometric Methods.

Hansbo, P. (2005). Nitsche's method for interface problems in computational mechanics. *GAMM-Mitteilungen*, 28(2):183–206.

Harari, I. and Grosu, E. (2015). A unified approach for embedded boundary conditions for fourth-order elliptic problems. *International Journal for Numerical Methods in Engineering*, 104(7):655–675.

Heinze, S., Bleistein, T., Düster, A., Diebels, S., and Jung, A. (2018). Experimental and numerical investigation of single pores for identification of effective metal foams properties. *Zeitschrift für Angewandte Mathematik und Mechanik*, 98:682–695.

Heinze, S., Joulaian, M., and Düster, A. (2015). Numerical homogenization of hybrid metal foams using the finite cell method. *Computers & Mathematics with Applications*, 70:1501–1517.

Hennig, P., Ambati, M., De Lorenzis, L., and Kästner, M. (2018). Projection and transfer operators in adaptive isogeometric analysis with hierarchical B-splines. *Computer Methods in Applied Mechanics and Engineering*, 334:313–336.

Hennig, P., Müller, S., and Kästner, M. (2016). Bézier extraction and adaptive refinement of truncated hierarchical NURBS. *Computer Methods in Applied Mechanics and Engineering*, 305:316–339.

Herrema, A. J., Johnson, E. L., Proserpio, D., Wu, M. C., Kiendl, J., and Hsu, M.-C. (2019). Penalty coupling of non-matching isogeometric KirchhoffLove shell patches with application to composite wind turbine blades. *Computer Methods in Applied Mechanics and Engineering*, 346:810–840.

Hiemstra, R. R., Sangalli, G., Tani, M., Calabrò, F., and Hughes, T. J. (2019). Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 355:234 – 260.

Hoang, T., Verhoosel, C. V., Auricchio, F., van Brummelen, E. H., and Reali, A. (2017). Mixed isogeometric finite cell methods for the stokes problem. *Computer Methods in Applied Mechanics and Engineering*, 316:400 – 423. Special Issue on Isogeometric Analysis: Progress and Challenges.

Hu, Q., Chouly, F., Hu, P., Cheng, G., and Bordas, S. P. (2018). Skew-symmetric nitsche's formulation in isogeometric analysis: Dirichlet and symmetry conditions, patch coupling and frictionless contact. *Computer Methods in Applied Mechanics and Engineering*, 341:188–220.

Hubrich, S. and Düster, A. (2019). Numerical integration for nonlinear problems of the finite cell method using an adaptive scheme based on moment fitting. *Computers & Mathematics with Applications*, 77(7):1983 – 1997.

Hubrich, S., Stolfo, P. D., Kudela, L., Kollmannsberger, S., Rank, E., Schröder, A., and Düster, A. (2017). Numerical integration of discontinuous functions: Moment fitting and smart octree. *Computational Mechanics*, pages 1–19.

Hug, L., Kollmannsberger, S., Yosibash, Z., and Rank, E. (2020). A 3D benchmark problem for crack propagation in brittle fracture. *Computer Methods in Applied Mechanics and Engineering*, 364:112905.

Hughes, T. J., Franca, L., Harari, I., Mallet, M., and Shakib, F. (1987). Finite element method for high-speed flows-consistent calculation of boundary flux. In *25th AIAA Aerospace Sciences Meeting*, page 556.

Hughes, T. J. R. (2000). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, Mineola, NY.

Hughes, T. J. R., Cottrell, J. A., and Bazilevs, Y. (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39–41):4135–4195.

Hughes, T. J. R., Engel, G., Mazzei, L., and Larson, M. G. (2000). The continuous galerkin method is locally conservative. *Journal of Computational Physics*, 163(2):467 – 488.

Igelbüscher, M., Schwarz, A., Steeger, K., and Schröder, J. (2019). Modified mixed least-squares finite element formulations for small and finite strain plasticity. *International Journal for Numerical Methods in Engineering*, 117:141–160.

ISO 10303-11:1994 (1994). Industrial automation systems and integration – Product data representation and exchange. Standard, International Organization for Standardization, Geneva, CH.

Jiang, W., Annavarapu, C., Dolbow, J. E., and Harari, I. (2015). A robust Nitsche's formulation for interface problems with spline-based finite elements. *International Journal for Numerical Methods in Engineering*, 104(7):676–696.

Johansson, A., Kehlet, B., Larson, M. G., and Logg, A. (2019). Multimesh finite element methods: Solving pdes on multiple intersecting meshes. *Computer Methods in Applied Mechanics and Engineering*, 343:672–689.

Jomo, J., de Prenter, F., Elhaddad, M., D'Angella, D., Verhoosel, C., Kollmannsberger, S., Kirschke, J., Nübel, V., van Brummelen, E., and Rank, E. (2019). Robust and parallel scalable iterative solutions for large-scale finite cell analyses. *Finite Elements in Analysis and Design*, 163:14–30.

Jomo, J., Zander, N., Elhaddad, M., Özcan, A., Kollmannsberger, S., Mundani, R.-P., and Rank, E. (2017). Parallelization of the multi-level hp-adaptive finite cell method. *Computers and Mathematics with Applications*, 74(1):126–142.

Joulaian, M., Duczek, S., Gabbert, U., and Düster, A. (2014). Finite and spectral cell method for wave propagation in heterogeneous materials. *Computational Mechanics*, 54(3):661–675.

Joulaian, M., Hubrich, S., and Düster, A. (2016). Numerical integration of discontinuities on arbitrary domains based on moment fitting. *Computational Mechanics*, 57(6):979–999.

Kamensky, D., Evans, J. A., Hsu, M.-C., and Bazilevs, Y. (2017). Projection-based stabilization of interface lagrange multipliers in immersogeometric fluidthin structure interaction analysis, with application to heart valve modeling. *Computers & Mathematics with Applications*, 74(9):2068–2088. Advances in Mathematics of Finite Elements, honoring 90th birthday of Ivo Babuka.

Kamensky, D. M. (2016). *Immersogeometric fluid–structure interaction analysis of bioprosthetic heart valves*. PhD thesis, The University of Texas at Austin.

Kiendl, J., Bletzinger, K.-U., Linhard, J., and Wüchner, R. (2009). Isogeometric shell analysis with Kirchhoff-Love elements. *Computer Methods in Applied Mechanics and Engineering*, 198(49):3902 – 3914.

Kohnke, P. (2009). Theory reference for the mechanical apdl and mechanical applications. *Ansys Inc, release*, 12.

Kollmannsberger, S., D'Angella, D., Rank, E., Garhuom, W., Hubrich, S., Düster, A., Stolfo, P. D., and Schröder, A. (2020). Spline- and hp-basis functions of higher differentiability in the finite cell method. *GAMM-Mitteilungen*, 43(1):e202000004.

Kollmannsberger, S., Özcan, A., Carraturo, M., Zander, N., and Rank, E. (2018). A hierarchical computational model for moving thermal loads and phase changes with applications to selective laser melting. *Computers & Mathematics with Applications*, 75(5):1483–1497.

Korelc, J. and Stupkiewicz, S. (2014). Closed-form matrix exponential and its application in finite-strain plasticity. *International Journal for Numerical Methods in Engineering*, 98:960–987.

Korshunova, N., Alaimo, G., Hosseini, S., Carraturo, M., Reali, A., Niiranen, J., Auricchio, F., Rank, E., and Kollmannsberger, S. (2021a). Image-based numerical characterization and experimental validation of tensile behavior of octet-truss lattice structures. *Additive Manufacturing*, 41:101949.

Korshunova, N., Alaimo, G., Hosseini, S. B., Carraturo, M., Reali, A., Niiranen, J., Auricchio, F., Rank, E., and Kollmannsberger, S. (2021b). Bending behavior of octet-truss lattice structures: modelling options, numerical characterization and experimental validation. *Materials & Design*.

Korshunova, N., Jomo, J., Lékó, G., Reznik, D., Balázs, P., and Kollmannsberger, S. (2020). Image-based material characterization of complex microarchitectured additively manufactured structures. *Computers and Mathematics with Applications*.

Kraft, R. (1997). Adaptive and linearly independent multilevel B-splines. In *Surface Fitting and Multiresolution Methods*. Vanderbilt University Press.

Kronbichler, M. and Kormann, K. (2019). Fast matrix-free evaluation of discontinuous galerkin finite element operators. *ACM Trans. Math. Softw.*, 45(3).

Kudela, L., Kollmannsberger, S., Almac, U., and Rank, E. (2020). Direct Structural Analysis of Domains Defined by Point Clouds. *Computer Methods in Applied Mechanics and Engineering*, 358.

Kudela, L., Zander, N., Bog, T., Kollmannsberger, S., and Rank, E. (2015). Efficient and accurate numerical quadrature for immersed boundary methods. *Advanced Modeling and Simulation in Engineering Sciences*, 2(1):1–22.

Kudela, L., Zander, N., Kollmannsberger, S., and Rank, E. (2016). Smart octrees: Accurately integrating discontinuous functions in 3D. *Computer Methods in Applied Mechanics and Engineering*, 306:406–426.

Lee, B.-G. and Park, Y. (2000). Degree elevation of B-spline curves and its matrix representation. *Journal of the Korean Society for Industrial and Applied Mathematics*, 4(2):1–9. 00001.

Li, X. and Scott, M. A. (2012). Analysis-suitable t-splines: characterization, refineability, and approximation. *CoRR*, abs/1211.5669.

Li, X., Zheng, J., Sederberg, T. W., Hughes, T. J., and Scott, M. A. (2012). On linear independence of t-spline blending functions. *Computer Aided Geometric Design*, 29(1):63–76. Geometric Constraints and Reasoning.

Loan, C. F. (2000). The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123(1):85 – 100. Numerical Analysis 2000. Vol. III: Linear Algebra.

Lorenzo, G., Scott, M. A., Tew, K., Hughes, T. J. R., and Gomez, H. (2017). Hierarchically refined and coarsened splines for moving interface problems, with particular application to phase-field models of prostate tumor growth. *Computer Methods in Applied Mechanics and Engineering*, 319:515–548.

Lyche, T. and Morken, K. (2008). Spline methods draft. www.uio.no/studier/emner/matnat/ifi/INF-MAT5340/v13/undervisningsmateriale/book.pdf, 2011 (Online; accessed 20.02.17).

Marussig, B. and Hughes, T. J. (2018). A review of trimming in isogeometric analysis: challenges, data exchange and simulation aspects. *Archives of computational methods in engineering*, 25(4):1059–1127.

MATLAB (2019). *version 9.6.0.1072779 (R2019a)*. The MathWorks Inc., Natick, Massachusetts.

Melbø, H. and Kvamsdal, T. (2003). Goal oriented error estimators for stokes equations based on variationally consistent postprocessing. *Computer Methods in Applied Mechanics and Engineering*, 192(5):613–633.

Melenk, J. M., Gerdes, K., and Schwab, C. (2001). Fully discrete hp-finite elements: Fast quadrature. *Computer Methods in Applied Mechanics and Engineering*, 190(32–33):4339–4364.

Müller, B., Kummer, F., and Oberlack, M. (2013). Highly accurate surface and volume integration on implicit domains by means of moment-fitting. *International Journal for Numerical Methods in Engineering*, 96(8):512–528.

Mungenast, M. (2017a). 3d-printed low-tech future facades - development of 3d-printed functional-geometries for building envelopes. In of Architecture, T. D. O. T. D. . F. and the Built Environment, editors, *January 19th 2017 - Munich Powerskin Conference*.

Mungenast, M. (2017b). Additive fertigung - anwendungsbeispiele für den modellbau. *DER ENTWURF Deutsche BauZeitschrift*.

Nagaraja, S., Elhaddad, M., Ambati, M., Kollmannsberger, S., De Lorenzis, L., and Rank, E. (2019). Phase-field modeling of brittle fracture with multi-level hp-FEM and the finite cell method. *Computational Mechanics*, 63(6):1283–1300.

Nguyena, V. P., Kerfriden, P., and Bordas, S. C. S. P. A. (2013). Nitsche's method method for mixed dimensional analysis: conforming and non-conforming continuum-beam and continuum-plate coupling.

Nitsche, J. (1971). Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36(1):9–15.

Orszag, S. A. (1980). Spectral methods for problems in complex geometries. *Journal of Computational Physics*, 37(1):70–92.

Oshima, M., Hughes, T. J., and Jansen, K. (1998). Consistent finite element calculations of boundary and internal fluxes. *International Journal of Computational Fluid Dynamics*, 9(3-4):227–235.

Özcan, A., Kollmannsberger, S., Jomo, J., and Rank, E. (2019). Residual stresses in metal deposition modeling: Discretizations of higher order. *Computers & Mathematics with Applications*, 78(7):2247–2266. Simulation for Additive Manufacturing.

Parvizian, J., Düster, A., and Rank, E. (2007). Finite cell method. *Computational Mechanics*, 41(1):121–133.

Parvizian, J., Düster, A., and Rank, E. (2011). Topology Optimization Using the Finite Cell Method. *Optimization and Engineering*, 13(1):57–78.

Piegl, L. and Tiller, W. (1995). *The NURBS Book*. Monographs in Visual Communications. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rank, E., Kollmannsberger, S., Sorger, C., and Düster, A. (2011). Shell Finite Cell Method: A high order fictitious domain approach for thin-walled structures. *Computer Methods in Applied Mechanics and Engineering*, 200(45-46):3200–3209.

Rank, E., Ruess, M., Kollmannsberger, S., Schillinger, D., and Düster, A. (2012). Geometric modeling, isogeometric analysis and the finite cell method. *Computer Methods in Applied Mechanics and Engineering*, 249-252:104–115.

Rudin, W. (1991). *Functional analysis*. McGraw-Hill Science, Engineering & Mathematics.

Ruess, M., Schillinger, D., Bazilevs, Y., Varduhn, V., and Rank, E. (2013). Weakly enforced essential boundary conditions for NURBS-embedded and trimmed NURBS geometries on the basis of the finite cell method. *International Journal for Numerical Methods in Engineering*, 95(10):811–846.

Ruess, M., Schillinger, D., Özcan, A. I., and Rank, E. (2014). Weak coupling for isogeometric analysis of non-matching and trimmed multi-patch geometries. *Computer Methods in Applied Mechanics and Engineering*, 269:46–71.

Ruess, M., Tal, D., Trabelsi, N., Yosibash, Z., and Rank, E. (2012). The finite cell method for bone simulations: verification and validation. *Biomechanics and Modeling in Mechanobiology*, 11(3):425–437.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition.

Salsa, S. (2016). *Equazioni a derivate parziali: Metodi, modelli e applicazioni*, volume 98. Springer.

Sangalli, G. and Tani, M. (2018). Matrix-free weighted quadrature for a computationally efficient isogeometric k-method. *Computer Methods in Applied Mechanics and Engineering*, 338:117 – 133.

Schillinger, D. (2012). *The P- and B-Spline Versions of the Geometrically Nonlinear Finite Cell Method and Hierarchical Refinement Strategies for Adaptive Isogeometric and Embedded Domain Analysis*. PhD thesis, Technische Universität München, Munich.

Schillinger, D., Dedè, L., Scott, M. A., Evans, J. A., Borden, M. J., Rank, E., and Hughes, T. J. (2012a). 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*, 249-252:116–150.

Schillinger, D., Düster, A., and Rank, E. (2012b). The Hp-D-Adaptive Finite Cell Method for Geometrically Nonlinear Problems of Solid Mechanics. *International Journal for Numerical Methods in Engineering*, 89(9):1171–1202.

Schillinger, D., Ruess, M., Zander, N., Bazilevs, Y., Düster, A., and Rank, E. (2012c). Small and large deformation analysis with the p- and B-spline versions of the Finite Cell Method. *Computational Mechanics*, 50(4):445–478.

Schillinger, D., Ruthala, P., and Nguyen, L. (2016). Lagrange extraction and projection for NURBS basis functions: A direct link between isogeometric and standard nodal finite element formulations: Lagrange extraction and projection for NURBS basis functions: A direct link between isogeometric and standard nodal finite element formulations. *International Journal for Numerical Methods in Engineering*, pages n/a–n/a.

Schillinger, D., Scott, M. A., Evans, J. A., Borden, M. J., Dede', L., Hughes, T. J., and Rank, E. (2012d). Isogeometric Analysis and the Finite Cell method. *Proceedings of 6th European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2012*, pages 6781–6791.

Scott, M., Li, X., Sederberg, T., and Hughes, T. J. R. (2012). Local refinement of analysis-suitable T-splines. *Computer Methods in Applied Mechanics and Engineering*, 213-216:206–222.

Scott, M., Thomas, D., and Evans, E. (2014). Isogeometric spline forests. *Computer Methods in Applied Mechanics and Engineering*, 269:222–264.

Scott, M. A., Borden, M. J., Verhoosel, C. V., Sederberg, T. W., and Hughes, T. J. R. (2011). Isogeometric finite element data structures based on Bézier extraction of T-splines. *International Journal for Numerical Methods in Engineering*, 88:126–156.

Siemens PLM Software Inc (2014). Nx nastran user's guide. https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/10/help/en_US/tdocExt/pdf/User.pdf. Accessed: 15-4-2021.

Simo, J. and Hughes, T. (1998). *Computational Inelasticity*. Interdisciplinary applied mathematics. Springer.

Simo, J. C. and Miehe, C. (1992). Associative coupled thermoplasticity at finite strains: Formulation, numerical analysis and implementation. *Computer Methods in Applied Mechanics and Engineering*, 98:41–104.

Stolfo, P. D., Rademacher, A., and Schröder, A. (2019). Dual weighted residual error estimation for the finite cell method. *Journal of Numerical Mathematics*, 27(2):101–122.

Strang, G. (1973). *An Analysis of the Finite Element Method*. Prentice-Hall, Englewood Cliffs, N.J.

Szabó, B. A. and Babuška, I. (1991). *Finite Element Analysis*. John Wiley & Sons, New York.

Szabó, B. A. and Babuška, I. (2011). *Introduction to Finite Element Analysis: Formulation, Verification, and Validation*. Wiley, Chichester, West Sussex.

Taghipour, A., Parvizian, J., Heinze, S., and Düster, A. (2018). The finite cell method for nearly incompressible finite strain plasticity problems with complex geometries. *Computers & Mathematics with Applications*, 75:3298–3316.

Thomas, D., Scott, M., Evans, J., Tew, K., and Evans, E. (2015). Bézier projection: A unified approach for local projection and quadrature-free refinement and coarsening of nurbs and t-splines with particular application to isogeometric design and analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:55 – 105. Isogeometric Analysis Special Issue.

Utku, M. and Carey, G. (1982). Boundary penalty techniques. *Computer Methods in Applied Mechanics and Engineering*, 30(1):103–118.

van Brummelen, E. H., van der Zee, K. G., Garg, V. V., and Prudhomme, S. (2011). Flux Evaluation in Primal and Dual Boundary-Coupled Problems. *Journal of Applied Mechanics*, 79(1).

Verhoosel, C., van Zwieten, G., van Rietbergen, B., and de Borst, R. (2015). Image-based goal-oriented adaptive isogeometric analysis with application to the micro-mechanical modeling of trabecular bone. *Computer Methods in Applied Mechanics and Engineering*, 284:138 – 164. Isogeometric Analysis Special Issue.

Vuong, A.-V. (2014). Finite Element Concepts and Bezier Extraction in Hierarchical Isogeometric Analysis. In *Progress in Industrial Mathematics at ECMI 2012*, pages 385–390. Springer, Cham.

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

Wahlbin, L. (1995). *Superconvergence in Galerkin Finite Element Methods*. Springer, New York, 1995 edition.

Wassermann, B., Kollmannsberger, S., Bog, T., and Rank, E. (2017). From geometric design to numerical analysis: A direct approach using the Finite Cell Method on Constructive Solid Geometry. *Computers & Mathematics with Applications*, 74(7):1703–1726. High-Order Finite Element and Isogeometric Methods 2016.

Wassermann, B., Kollmannsberger, S., Yin, S., Kudela, L., and Rank, E. (2019). Integrating CAD and numerical analysis: 'Dirty geometry' handling using the Finite Cell Method. *Computer Methods in Applied Mechanics and Engineering*, 351:808–835.

Wriggers, P. and Hudobivnik, B. (2017). A low order virtual element formulation for finite elasto-plastic deformations. *Computer Methods in Applied Mechanics and Engineering*, 327:459–477.

Wu, M. C., Kamensky, D., Wang, C., Herrema, A. J., Xu, F., Pigazzini, M. S., Verma, A., Marsden, A. L., Bazilevs, Y., and Hsu, M.-C. (2017). Optimizing fluidstructure interaction systems with immersogeometric analysis and surrogate modeling: Application to a hydraulic arresting gear. *Computer Methods in Applied Mechanics and Engineering*, 316:668–693. Special Issue on Isogeometric Analysis: Progress and Challenges.

Yang, Z., Kollmannsberger, S., Düster, A., Ruess, M., Garcia, E. G., Burgkart, R., and Rank, E. (2012a). Non-Standard Bone Simulation: Interactive Numerical Analysis by Computational Steering. *Computing and Visualization in Science*, 14(5):207–216.

Yang, Z., Ruess, M., Kollmannsberger, S., Düster, A., and Rank, E. (2012b). An Efficient Integration Technique for the Voxel-Based Finite Cell Method. *International Journal for Numerical Methods in Engineering*, 91(5):457–471.

Zander, N. (2017). *Multi-Level Hp-FEM: Dynamically Changing High-Order Mesh Refinement with Arbitrary Hanging Nodes*. Ph.D. Thesis, Technische Universität München, München.

Zander, N., Bog, T., Elhaddad, M., Frischmann, F., Kollmannsberger, S., and Rank, E. (2016). The multi-level hp-method for three-dimensional problems: Dynamically changing high-order mesh refinement with arbitrary hanging nodes. *Computer Methods in Applied Mechanics and Engineering*, 310:252–277.

Zander, N., Bog, T., Kollmannsberger, S., Schillinger, D., and Rank, E. (2015). Multi-level hp-adaptivity: High-order mesh adaptivity without the difficulties of constraining hanging nodes. *Computational Mechanics*, 55(3):499–517.

Zander, N., Kollmannsberger, S., Ruess, M., Yosibash, Z., and Rank, E. (2012). The Finite Cell Method for linear thermoelasticity. *Computers & Mathematics with Applications*, 64(11):3527–3541.

Zander, N., Ruess, M., Bog, T., Kollmannsberger, S., and Rank, E. (2017). Multi-Level hp-Adaptivity for Cohesive Fracture Modeling. *International Journal for Numerical Methods in Engineering*, 109(13):1723–1755.