POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Non-conforming mesh adaptivity for Hybrid High-Order methods

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Alessandra Crippa**

Student ID: 939781
Advisor: Prof. Paola Antonietti
Co-advisors: Prof. Daniele Di Pietro
Academic Year: 2021-22

Abstract

The Hybrid High-Order (HHO) methods are discretization methods for partial differential equations which have some advantages with respect to traditional numerical schemes; in particular they support polytopal meshes and arbitrary approximation orders. The elements of the mesh can have arbitrary shape and can coexist in the same mesh with different shapes and number of faces. A direct consequence is the possibility to discretize the physical domain through non-conforming meshes, which is particularly relevant for physical phenomena occurring in small areas of the domain. In this case is indeed possible to locally refine the mesh, without reconstructing it in the entire domain. As a result, a significant advantage in terms of computational cost is achieved.

After introducing the HHO method for the Poisson problem, the goal of this work will be to present a method to perform non-conforming mesh adaptivity, exploiting a-posteriori estimators.

Starting from the HArD::Core library for HHO methods, developed by D. Di Pietro and J. Droniou and written in C++, new methods for a non-conforming adaptivity of the mesh have been developed, which are included in the new class DynamicMesh. First the implementation aspects will be presented, then some numerical tests with a non-conforming refined mesh will be performed.

Keywords: Polytopal methods, Hybrid High-Order methods, A posteriori error estimators, Non-conforming mesh refinement

Estratto

I metodi Hybrid High-Order (HHO) sono metodi per la discretizzazione di equazioni alle differenze parziali che comportano alcuni vantaggi rispetto ai metodi numerici tradizionali; in particolare supportano mesh politopali e ordini di approssimazione arbitrari. Gli elementi della mesh possono avere forma qualsiasi e possono essere contemporaneamente presenti nella stessa griglia con forma e numero di lati differenti. Conseguenza diretta è la possibilità di discretizzare il dominio fisico tramite mesh non conformi, il che risulta particolarmente utile nel caso di fenomeni fisici concentrati in un punto o un'area ristretta del dominio, poichè diventa possibile raffinare la mesh solo localmente, senza doverla ricostruire nell'intero dominio. Ne risulta un guadagno non indifferente a livello di costo computazionale.

In questo lavoro, dopo aver introdotto il metodo HHO per il problema di Poisson, lo scopo sarà presentare un metodo per eseguire adattività della mesh non conforme, sfruttando idealmente stimatori a posteriori dell'errore.

Partendo dalla libreria HArD::Core per metodi HHO, realizzata da D. Di Pietro e J. Droniou in C++, sono stati sviluppati nuovi metodi - da integrare successivamente alla libreria - contenuti nella nuova classe DynamicMesh, in grado di supportare un'adattività non conforme della mesh. Verranno presentati dapprima gli aspetti implementativi della nuova classe, ed infine alcuni test numerici con mesh raffinata in maniera non conforme.

Parole chiave: Metodi politopali, Metodi Hybrid High-Order, Stimatori a posteriori, Raffinamento non conforme della mesh.

Contents

Introduction

Hybrid High-order methods are discretization methods for partial differential equations belonging to a family of numerical schemes which has recently undergone a flood of interest, namely the polytopal methods. HHO are hybrid methods, namely they support unknowns attached to both faces and cells, and can have arbitrary approximation orders. Moreover, the crucial feature of polytopal methods is the freedom they give in discretizing the physical domain. They can indeed easily handle hanging nodes, non-matching interfaces, and elements with different shapes and number of faces in the same mesh. As a result non-conforming meshes can come into play and flexibly be used to discretize complex domains and interfaces, as well as to deal phyisical phenomena occurring in one point or in a small area of the domain, allowing a local adaptation of the mesh. The latter cleary results in a significant computational cost improvement, making unnecessary to re-build the mesh in the entire domain to preserve the quality of the mesh.

A local non-conforming adaptivity of the mesh is the main focus of this work, whose goal will be, after introducing the HHO methods, to apply to them an automatic procedure for a local non-conforming refinement of the mesh. We will show that in this way we will be able to recover optimal order of convergence of the numerical solution also in case of poorly regular exact solution, whereas with a uniformly refined sequence of meshes the convergence rate is limited by the irregularities.
It is desirable for a proper adaptation procedure to be driven by suitable a posteriori error estimators; however for now we limit our work only to adaptivity driven by the true error, exploiting benchmark cases with known solutions.

In Chapter 1 we introduce the HHO method for the Poisson problem with Dirichlet boundary conditions, starting by the discrete setting needed for the method and proceeding then with the key elements for the discretization of the problem. We will show some error estimates, valid under appropriate regularity hypothesis for the exact solution, and finally we briefly report how the method is numerically implemented, highlighting how HHO methods become skeletal methods through the crucial technique of *static conden-*

sation, which allows to reduce the degrees of freedom of the problem accounting only for face unknowns.
In Chapter 2 we give a brief overview of residual-based error estimators, that provide reliable, fully computable and efficient upper bounds on the error. As a result they are suitable to drive mesh adaptivity procedures, as we will show in upcoming work.
In Chapter 3 we present our original contribution to the work, namely some methods to perform non-conforming adaptivity of the mesh, in the context of HArD::Core library, a suite of C++ tools for polytopal methods, in particular HHO methods, developed by D. Di Pietro and J.Droniou, which still lacked the functions to perform a non conforming refinement and coearsening of the mesh. Finally we test the implemented method on a benchmark case with a known irregular solution, showing that, through a non-conforming adaptation procedure, we are able to recover optimal order of convergence of the numerical solution.

1 | The HHO method

In this chapter we present the HHO method for the Poisson problem with Dirichlet boundary conditions, starting with the necessary discrete setting, then proceeding with the key elements of the numerical scheme. We show some error estimates and briefly explain how the method is implemented. The discussion references to [2], [3], [6] and [5].

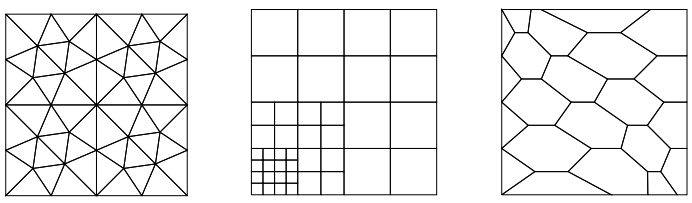
1.1. Discrete setting

In this section we expose some definitions and conventions that will be used in the rest of the presentation. Starting from the notion of polytopal mesh of Ω, we formulate assumptions on the way meshes are refined, and we introduce some local functional spaces and projectors that will be needed in the construction and analysis of HHO methods.

1.1.1. Polytopal mesh

As we said (da scrivere intro) the support of polytopal meshes with possibly non-matching interfaces is one of the key features of HHO methods, allowing us to perform local non-conforming mesh adaptivity.
Let us start with some useful definitions.

Definition 1.1 (Simplex and polytopal set)**.** Let an integer $d \geq 2$ be fixed. Given a set of *vertices* $\mathcal{P} \coloneqq \{\boldsymbol{P}_0, \dots, \boldsymbol{P}_d\} \subset \mathbb{R}^d$ such that the family of vectors $\{\boldsymbol{P}_1 - \boldsymbol{P}_0, \dots, \boldsymbol{P}_d - \boldsymbol{P}_0\}$



(a) Matching triangular (b) Non-conforming (c) Polygonal

Figure 1.1: Examples of polytopal meshes in two and three space dimensions, from [2].

is linearly independent, the interior of the convex hull of $\mathcal{P}$ is a *simplex* of $\mathbb{R}^d$. For each integer $i \in \{0, \dots, d\}$, the convex hull of $\mathcal{P} \setminus \{\boldsymbol{P}_i\}$ is a *simplicial face*. A polytopal set (or polytope) is a connected set that is the interior of a finite union of closures of simplices.

This means that a simplex is an open triangle (respectively a tethraedron) in dimension d=2 (respectively d=3), and a polytope is an open polygonal (respectively a polyhedral set).

From now on the domain Ω of our interest will be a - bounded, open, connected - polytopal set of $\mathbb{R}^d$.

The previous definition brings us to the following one.

Definition 1.2 (Polytopal mesh)**.** A *polytopal mesh* of Ω is a couple $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ where:

(i) The set of *mesh elements* $\mathcal{T}_h$ is a finite collection of nonempty disjoint polytopes T with boundary ∂T and diameter h_T such that the *meshsize* h satisfies

$$h = \max_{T \in \mathcal{T}_h} h_T$$

and it holds

$$\overline{\Omega} = \bigcup_{T \in \mathcal{T}_h} \overline{T}.$$

(ii) The set of *mesh faces* $\mathcal{F}_h$ is a finite collection of disjoint subsets of $\overline{\Omega}$ such that, for any $F \in \mathcal{F}_h$, F is a non-empty open connected subset of a hyperplane of $\mathbb{R}^d$ and the $(d-1)$-dimensional Hausdorff measure of its relative boundary $\overline{F} \setminus F$ is zero. We denote by h_F the diameter of F. Further assume that:

(a) For each $F \in \mathcal{F}_h$, either there exist distinct mesh elements $T_1, T_2 \in \mathcal{T}_h$ such that $F \subset \partial T_1 \cap \partial T_2$ and F is called an *interface*, or there exists one mesh element $T \in \mathcal{T}_h$ such that $F \subset \partial T \cap \partial\Omega$ and F is called a *boundary face*;

(b) The set of mesh faces is a partition of the mesh skeleton, i.e.,

$$\bigcup_{T \in \mathcal{T}_h} \partial T = \bigcup_{F \in \mathcal{F}_h} \overline{F}.$$

Interfaces are collected in the set $\mathcal{F}_h^{\mathrm{i}}$ and boundary faces in $\mathcal{F}_h^{\mathrm{b}}$, so that $\mathcal{F}_h = \mathcal{F}_h^{\mathrm{i}} \cup \mathcal{F}_h^{\mathrm{b}}$.

For any mesh element $T \in \mathcal{T}_h$,

$$\mathcal{F}_T \coloneqq \{F \in \mathcal{F}_h \ : \ F \subset \partial T\}$$

denotes the set of faces contained in ∂T. Symmetrically, for any mesh face $F \in \mathcal{F}_h$,

$$\mathcal{T}_F \coloneqq \{T \in \mathcal{T}_h \ : \ F \subset \partial T\} \tag{1.1}$$

is the set containing the one or two mesh elements sharing F. Finally, for all $T \in \mathcal{T}_h$ and all $F \in \mathcal{F}_T$, $\boldsymbol{n}_{TF}$ denotes the unit normal vector to F pointing out of T.

1.1.2. Regular mesh sequences

In order to study the convergence of HHO methods with respect to the meshsize h we need to make regular assumptions on how the mesh is refined. Here we refer to *isotropic meshes* with *non-degenerate faces*.

Definition 1.3 (Matching Simplicial Mesh)**.** $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ is a *simplicial mesh* of Ω if, for all $T \in \mathcal{T}_h$, T is a simplex of $\mathbb{R}^d$. $\mathcal{M}_h$ is a *matching simplicial mesh* of Ω if it is a simplicial mesh and the following additional conditions hold: (i) For any $T, T' \in \mathcal{T}_h$ with $T' \neq T$, the set $\partial T \cap \partial T'$ is the convex hull of a (possibly empty) subset of the vertices of T; (ii) The set $\mathcal{F}_h$ is composed of the simplicial faces of the elements in $\mathcal{T}_h$.

Definition 1.4 (Matching simplicial submesh)**.** Let $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ be a polytopal mesh of Ω. We say that $\mathfrak{M}_h = (\mathfrak{T}_h, \mathfrak{F}_h)$ is a *matching simplicial submesh* of $\mathcal{M}_h$ if: (i) $\mathfrak{M}_h$ is a matching simplicial mesh of Ω; (ii) for any simplex $\tau \in \mathfrak{T}_h$, there is a unique mesh element $T \in \mathcal{T}_h$ such that $\tau \subset T$; (iii) for any simplicial face $\sigma \in \mathfrak{F}_h$ and any mesh face $F \in \mathcal{F}_h$, either $\sigma \cap F = \emptyset$ or $\sigma \subset F$.

The notion of matching simplicial submesh (Fig. 1.2) is merely theoretical: it will not be constructed in practice.

Definition 1.5 (Regular mesh sequence)**.** Denote by $\mathcal{H} \subset (0, +\infty)$ a countable set of meshsizes having 0 as its unique accumulation point. A family of meshes $(\mathcal{M}_h)_{h\in\mathcal{H}} = (\mathcal{T}_h, \mathcal{F}_h)_{h\in\mathcal{H}}$ is said to be *regular* if there exists a real number $\varrho \in (0, 1)$, independent of h and called the *mesh regularity parameter*, such that, for all $h \in \mathcal{H}$, there exists a matching simplicial submesh $\mathfrak{M}_h = (\mathfrak{T}_h, \mathfrak{F}_h)$ of $\mathcal{M}_h$ that satisfies the following conditions:

(i) *Shape regularity.* For any simplex $\tau \in \mathfrak{T}_h$, denoting by h_τ its diameter and by r_τ its

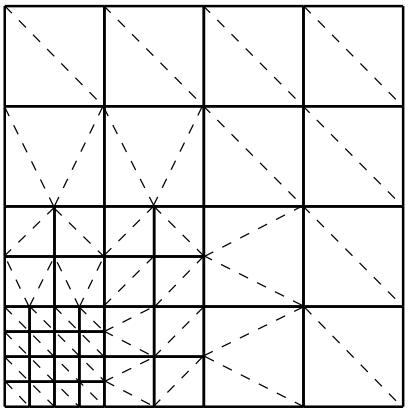


Figure 1.2: Example of a matching simplicial submesh (dashed lines) of the non-conforming mesh in Fig. (1.1b), taken from [2]

.

inradius, it holds

$$\varrho h_\tau \leq r_\tau; \tag{1.2}$$

(ii) *Contact regularity.* For any mesh element $T \in \mathcal{T}_h$ and any simplex $\tau \in \mathfrak{T}_T$, where $\mathfrak{T}_T \coloneqq \{\tau \in \mathfrak{T}_h \ : \ \tau \subset T\}$ is the set of simplices contained in T, it holds

$$\varrho h_T \leq h_\tau. \tag{1.3}$$

1.1.3. Local and broken polynomial spaces

Let $\mathcal{M}_h$ denote a polytopal mesh of Ω. We define the broken Sobolev space

$$W^{s,p}(\mathcal{T}_h) \coloneqq \left\{v \in L^p(\Omega) \ : \ v_{|T} \in W^{s,p}(T) \quad \forall T \in \mathcal{T}_h\right\}.$$

Remark. Functions in $W^{1,p}(\mathcal{T}_h)$ in general do not admit a global weak gradient. We can, however, define the broken gradient operator $\boldsymbol{\nabla}_h : W^{1,p}(\mathcal{T}_h) \to L^p(\Omega)^d$ such that, for all $v \in W^{1,p}(\mathcal{T}_h)$,

$$(\boldsymbol{\nabla}_h v)_{|T} \coloneqq \boldsymbol{\nabla}(v_{|T}) \qquad \forall T \in \mathcal{T}_h. \tag{1.4}$$

We denote the space of n-variate poynomials of total degree l as

$$\mathbb{P}_n^l \coloneqq \Bigg\{ p : \mathbb{R}^n \to \mathbb{R} \ : \exists (\gamma_{\boldsymbol{\alpha}})_{\boldsymbol{\alpha} \in \boldsymbol{A}_n^l} \in \mathbb{R}^{N_n^l} \text{ such that } p(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha} \in \boldsymbol{A}_n^l} \gamma_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \Bigg\},$$

with

$$\boldsymbol{A}_n^s \coloneqq \{ \boldsymbol{\alpha} \in \mathbb{N}^n \ : \ \|\boldsymbol{\alpha}\|_1 \leq s \}, \tag{1.5}$$

$$N_n^l \coloneqq \operatorname{card}(\boldsymbol{A}_n^l) = \binom{l+n}{n}, \tag{1.6}$$

and, for a given multi-index $\boldsymbol{\alpha} \in \boldsymbol{A}_n^l$, we have set

$$\boldsymbol{x}^{\boldsymbol{\alpha}} \coloneqq x_1^{\alpha_1} \cdots x_n^{\alpha_n}.$$

Definition 1.6 (Local polynomial space)**.** Let $X \subset \mathbb{R}^n$, $n \geq 1$, be an open bounded connected set, and let an integer $l \geq 0$ be fixed. The *local (real-valued) polynomial space* $\mathbb{P}^l(X)$ is defined as the space spanned by the restrictions to X of functions in the polynomial space $\mathbb{P}_n^l$.

More generally, if V is a finite-dimensional vector space, the *V-valued local polynomial space* $\mathbb{P}^l(X;V)$ is the space of functions $f : X \to V$ such that the components of f on a basis of V belong to $\mathbb{P}^l(X)$; we note that this definition does not depend on the chosen basis (if the components in one basis are polynomial, then the components in any basis are polynomial).

Definition 1.7 (Broken polynomial space)**.** Let $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ denote a polytopal mesh of Ω in the sense of Definition 1.2, and let an integer $l \geq 0$ be given. We define the *broken polynomial space*

$$\mathbb{P}^l(\mathcal{T}_h) \coloneqq \left\{ v_h \in L^1(\Omega) \ : \ v_{h|T} \in \mathbb{P}^l(T) \quad \forall T \in \mathcal{T}_h \right\}.$$

1.1.4. Projectors on local polynomial spaces

In the design and analysis of HHO methods a key role is played by projectors on local polynomial spaces. We are in fact interested in *projecting* functions on each element of our mesh.

Definition 1.8 (Projector on a local polynomial space)**.** Let an integer $l \geq 0$ and an

open bounded connected set $X \subset \mathbb{R}^n$, $n \geq 1$, be given. Let W be a vector space such that $\mathbb{P}^l(X) \subset W$. A linear mapping $\Pi_X^l : W \to \mathbb{P}^l(X)$ is a *projector on the local polynomial space* $\mathbb{P}^l(X)$ if it is onto and idempotent, i.e., $\Pi_X^l \circ \Pi_X^l = \Pi_X^l$.

Projectors on local polynomial spaces are characterised by the property of polynomial invariance.

Proposition 1.1 (Characterisation of projectors on local polynomial spaces)**.** *Let an integer $l \geq 0$ and an open bounded connected set $X \subset \mathbb{R}^n$, $n \geq 1$, be given. Let W be a vector space such that $\mathbb{P}^l(X) \subset W$. A linear mapping $\Pi_X^l : W \to \mathbb{P}^l(X)$ is a projector on the local polynomial space $\mathbb{P}^l(X)$ in the sense of Definition 1.8 if and only if, for any $v \in \mathbb{P}^l(X)$,*

$$\Pi_X^l v = v. \tag{1.7}$$

Proof. Let us assume that Π_X^l is onto and idempotent, and let us prove (1.7). Take $v \in \mathbb{P}^l(X)$. Since Π_X^l is onto, there exists $w \in W$ such that $v = \Pi_X^l w$. Taking the projection of this equality and using the idempotence property, we obtain

$$\Pi_X^l v = \Pi_X^l(\Pi_X^l w) = \Pi_X^l w = v,$$

which is (1.7).

Assume now (1.7). Then, since $\mathbb{P}^l(X) \subset W$, we have that

$$\mathbb{P}^l(X) = \Pi_X^l \mathbb{P}^l(X) \subset \Pi_X^l W \subset \mathbb{P}^l(X),$$

which shows that $\Pi_X^l W = \mathbb{P}^l(X)$, i.e., Π_X^l is onto. Moreover, using again the polynomial invariance (1.7) we have, for any $w \in W$, that $\Pi_X^l(\Pi_X^l w) = \Pi_X^l w$, which proves that Π_X^l is idempotent. □

The L^2 orthogonal projector

One of the two fundamental projectors on local polynomial spaces we will need to carry out the construction of the HHO methods is the following.

Definition 1.9 (The L^2-orthogonal projector)**.** The L^2-orthogonal projector (in short, L^2-projector) $\pi_X^{0,l} : L^1(X) \to \mathbb{P}^l(X)$ is defined as follows: For all $v \in L^1(X)$, the polynomial $\pi_X^{0,l} v \in \mathbb{P}^l(X)$ satisfies

$$(\pi_X^{0,l} v - v, w)_X = 0 \qquad \forall w \in \mathbb{P}^l(X). \tag{1.8}$$

Riesz representation theorem in $\mathbb{P}^l(X)$ for the standard $L^2(X)$-inner product entails existence and uniqueness.

Moreover, the following characterization can be proved:

$$\pi_X^{0,l} v = \operatorname*{argmin}_{w \in \mathbb{P}^l(X)} \|w - v\|_X^2.$$

It is easy to check that $\pi_X^{0,l}$ satisfies (1.7), i.e. it is polynomial invariant. Indeed, if $v \in \mathbb{P}^l(X)$, then $w = \pi_X^{0,l} v - v \in \mathbb{P}^l(X)$; hence, from (1.8),

$$(\pi_X^{0,l} v - v, \pi_X^{0,l} v - v)_X = 0.$$

As consequence

$$\pi_X^{0,l} v = v.$$

Let's introduce now also the *global L^2-orthogonal projectors on broken spaces*

$$\pi_h^{0,l} : L^1(\Omega) \to \mathbb{P}^l(\mathcal{T}_h),$$

such that for all $v \in L^1(\Omega)$ and all $T \in \mathcal{T}_h$,

$$(\pi_h^{0,l} v)_{|T} = \pi_T^{0,l} v_{|T}. \tag{1.9}$$

The elliptic projector

The other key example of projectors on local polynomial spaces is the following.

Definition 1.10 (The elliptic projector)**.** The elliptic projector $\pi_X^{1,l} : W^{1,1}(X) \to \mathbb{P}^l(X)$ is defined as follows: For all $v \in W^{1,1}(X)$, the polynomial $\pi_X^{1,l} v \in \mathbb{P}^l(X)$ satisfies

$$(\boldsymbol{\nabla}(\pi_X^{1,l} v - v), \boldsymbol{\nabla} w)_X = 0 \qquad \forall w \in \mathbb{P}^l(X) \tag{1.10a}$$

and

$$(\pi_X^{1,l} v - v, 1)_X = 0. \tag{1.10b}$$

By the Riesz representation theorem in $\boldsymbol{\nabla}\mathbb{P}^l(X)$ for the $L^2(X)^n$-inner product, (1.10a) defines a unique element $\boldsymbol{\nabla}\pi_X^{1,l} v \in \boldsymbol{\nabla}\mathbb{P}^l(X)$, and thus a polynomial $\pi_X^{1,l} v$ up to an additive constant. This constant is fixed by (1.10b).

The following characterisation can be proven:

$$\pi_X^{1,l} v = \underset{w\in\mathbb{P}^l(X),\,(w-v,1)_X=0}{\operatorname{argmin}} \|\boldsymbol{\nabla}(w-v)\|_X^2.$$

In order to check if $\pi_X^{1,l}$ satisfies the condition of polynomial invariance (1.7), let us consider $v \in \mathbb{P}^l(X)$ and $w = \pi_X^{1,l} v - v \in \mathbb{P}^l(X)$. Then, by (1.10a), $\boldsymbol{\nabla}(\pi_X^{1,l} v - v) = \mathbf{0}$. It follows that $\pi_X^{1,l} v$ and v only differ by a constant, which must be zero due to (1.10b).

Theorem 1.1 (Approximation properties of the L^2-orthogonal and elliptic projectors)**.** *Let $(\mathcal{M}_h)_{h\in\mathcal{H}}$ be a regular mesh sequence. Let a polynomial degree $l \geq 0$, an integer $s \in \{0,\dots,l+1\}$, and a real number $p \in [1,\infty]$ be given. Then, for any X element or face of $\mathcal{M}_h$, all $v \in W^{s,p}(X)$, and all $m \in \{0,\dots,s\}$,*

$$|v - \pi_X^{0,l} v|_{W^{m,p}(X)} \lesssim h_X^{s-m} |v|_{W^{s,p}(X)}. \tag{1.11}$$

and

$$|v - \pi_T^{1,l} v|_{W^{m,p}(T)} \lesssim h_T^{s-m} |v|_{W^{s,p}(T)}. \tag{1.12}$$

1.2. Basic principles of Hybrid High-Order methods: Poisson problem

In this section we present the HHO methods applied to the Poisson problem:
Find $u : \Omega \to \mathbb{R}$ such that

$$-\Delta u = f \qquad \text{in } \Omega, \tag{1.13a}$$

$$u = 0 \qquad \text{on } \partial\Omega, \tag{1.13b}$$

where Ω is an open bounded polytopal subset of $\mathbb{R}^n$, $n \geq 2$, with boundary $\partial\Omega$ and $f : \Omega \to \mathbb{R}$ is a given volumetric source term, assumed to be in $L^2(\Omega)$.
Its weak formulation is the following:
Find $u \in H_0^1(\Omega)$ such that

$$a(u,v) = (f,v) \qquad \forall v \in H_0^1(\Omega), \tag{1.14}$$

where the bilinear form $a : H^1(\Omega) \times H^1(\Omega) \to \mathbb{R}$ is such that

$$a(u,v) \coloneqq (\boldsymbol{\nabla} u, \boldsymbol{\nabla} v). \tag{1.15}$$

1.2.1. Local construction

In this section we present the key local ingredients for developing the HHO methods:

- the *discrete unknowns*;
- the *interpolator*;
- the *potential reconstruction operator*;
- the *local approximation of the continuous bilinear form.*

The discrete unknowns of the HHO methods are defined over elements and faces of a polytopal mesh. Let us look for suitable spaces for them. Let us start with an important remark, namely, let's see how it's possible to compute the elliptic projection starting from the L^2 orthogonal one. This result will inspire us to choose the right spaces for the discrete unknowns.

Given a function $v \in W^{1,1}(T)$ and a function $w \in C^\infty(\overline{T})$ the following integration by parts formula holds:

$$(\boldsymbol{\nabla} v, \boldsymbol{\nabla} w)_T = -(v, \Delta w)_T + \sum_{F \in \mathcal{F}_T} (v, \boldsymbol{\nabla} w \cdot \boldsymbol{n}_{TF})_F. \tag{1.16}$$

Considering in particular $w \in \mathbb{P}^{k+1}(T)$ we can observe that, since $\Delta w \in \mathbb{P}^{k-1}(T) \subset \mathbb{P}^k(T)$ and thanks to the Definition (1.8) of the L^2-orthogonal projector, $(v, \Delta w)_T = (\pi_T^{0,k} v, \Delta w)_T$. Analogously, applying the same Definition (1.8) since $(\boldsymbol{\nabla} w)_{|F} \cdot \boldsymbol{n}_{TF} \in \mathbb{P}^k(F)$, we can write $(v, \boldsymbol{\nabla} w \cdot \boldsymbol{n}_{TF})_F = (\pi_F^{0,k} v, \boldsymbol{\nabla} w \cdot \boldsymbol{n}_{TF})_F$ for all $F \in \mathcal{F}_T$. Finally, we can exploit the Definition (1.10) of elliptic projectors to write $(\boldsymbol{\nabla} v, \boldsymbol{\nabla} w)_T = (\boldsymbol{\nabla} \pi_T^{1,k+1} v, \boldsymbol{\nabla} w)_T$. To conclude, we obtain from (1.16) the equivalent relation:

$$(\boldsymbol{\nabla} \pi_T^{1,k+1} v, \boldsymbol{\nabla} w)_T = -(\pi_T^{0,k} v, \Delta w)_T + \sum_{F \in \mathcal{F}_T} (\pi_F^{0,k} v, \boldsymbol{\nabla} w \cdot \boldsymbol{n}_{TF})_F. \tag{1.17a}$$

Moreover, recalling once again the definitions of the projectors (1.10b) and (1.8), we get

$$0 = (\pi_T^{1,k+1} v - v, 1)_T = (\pi_T^{1,k+1} v - \pi_T^{0,k} v, 1)_T. \tag{1.17b}$$

From the above relation (1.17) we can observe that to compute the elliptic projection $\pi_T^{1,k+1} v$ we do not need the full knowledge of v but it is sufficient the knowledge of the L^2-orthogonal projections of v on $\mathbb{P}^k(T)$ and $\mathbb{P}^k(F)$ for all$F \in \mathcal{F}_T$.

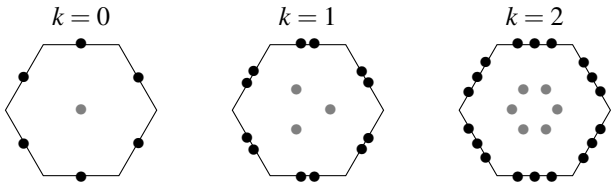


Figure 1.3: Discrete unknowns in $\underline{U}_T^k$ for $k \in \{0,1,2\}$. The dots represent the number of unknowns attached to an element or face, in dimension $d = 2$. Figure taken from [2].

The idea of the HHO method is therefore to write a scheme where the unknowns approximate $\pi_T^{0,k} v$ and $\pi_F^{0,k} v$.

Dicrete Unknowns

The last remark allows us to introduce the space of discrete unknowns the HHO construction hinges on (see Fig. 1.3):

$$\underline{U}_T^k \coloneqq \left\{ \underline{v}_T = (v_T, (v_F)_{F\in\mathcal{F}_T}) \ : \ v_T \in \mathbb{P}^k(T) \text{ and } v_F \in \mathbb{P}^k(F) \quad \forall F \in \mathcal{F}_T \right\}, \tag{1.18}$$

endowed with the H^1-like seminorm $\|\cdot\|_{1,T}$ such that, for all $\underline{v}_T \in \underline{U}_T^k$,

$$\begin{aligned} \|\underline{v}_T\|_{1,T} &\coloneqq \left(\|\boldsymbol{\nabla} v_T\|_T^2 + |\underline{v}_T|_{1,\partial T}^2 \right)^{\frac{1}{2}}, \\ |\underline{v}_T|_{1,\partial T} &\coloneqq \left(\sum_{F\in\mathcal{F}_T} h_F^{-1} \|v_F - v_T\|_F^2 \right)^{\frac{1}{2}}, \end{aligned} \tag{1.19}$$

where h_F is the diameter of F. The negative power of h_F in the boundary term $|\underline{v}_T|_{1,\partial T}$ has been chosen in order to make both terms in the seminorm homogeneous from the point of view of the dimension.

The idea is that the discrete unknowns represent projections L^2 of degree k on the elements and faces of the mesh. This leads us to introduce the following.

Local Interpolator

The second central component in our treatment is the local interpolator $\underline{I}_T^k$, which allows to obtain the discrete unknowns corresponding to a smooth function $v \in W^{1,1}(T)$.

$$\underline{I}_T^k : W^{1,1}(T) \to \underline{U}_T^k, \qquad \underline{I}_T^k v \coloneqq (\pi_T^{0,k} v, (\pi_F^{0,k} v)_{F\in\mathcal{F}_T}). \tag{1.20}$$

Once we have defined the local interpolator we have that the unknowns of the space HHO

represent exactly the projections L^2 of degree k on elements and faces of the mesh.

Proposition 1.2 (Boundedness of the local interpolator). *For all $T \in \mathcal{T}_h$ and all $v \in H^1(T)$,*

$$\|\underline{I}_T^k v\|_{1,T} \lesssim |v|_{H^1(T)}, \tag{1.21}$$

where the hidden constant depends only on d, ϱ and k.

Potential Reconstructor Operator

We now proceed, inspired by (1.17) (i.e. trying to replicate it), introducing the heart of the HHO method, namely the potential reconstruction operator. $\mathrm{p}_T^{k+1} : \underline{U}_T^k \to \mathbb{P}^{k+1}(T)$, such that, for all $\underline{v}_T \in \underline{U}_T^k$,

$$(\boldsymbol{\nabla}\mathrm{p}_T^{k+1}\underline{v}_T, \boldsymbol{\nabla}w)_T = -(v_T, \Delta w)_T + \sum_{F\in\mathcal{F}_T}(v_F, \boldsymbol{\nabla}w\cdot\boldsymbol{n}_{TF})_F \quad \forall w \in \mathbb{P}^{k+1}(T) \tag{1.22a}$$

and

$$(\mathrm{p}_T^{k+1}\underline{v}_T - v_T, 1)_T = 0. \tag{1.22b}$$

Note that (1.22a) univocally defines the gradient of p_T^{k+1}, so that, in order to obtain a unique reconstruction $\mathrm{p}_T^{k+1}\underline{v}_T \in \mathbb{P}^{k+1}(T)$, we need to add the second condition (1.22b) on its mean value.

Note moreover that, for all $v \in W^{1,1}(T)$,

$$\mathrm{p}_T^{k+1}\underline{I}_T^k v = \pi_T^{1,k+1} v. \tag{1.23}$$

This property shows that, taken v sufficiently regular, the potential reconstruction applied to the interpolator of v is an optimal approximation of v in the polynomial space $\mathbb{P}^{k+1}(T)$ and this fact will be useful later to obtain the consistency of the method.

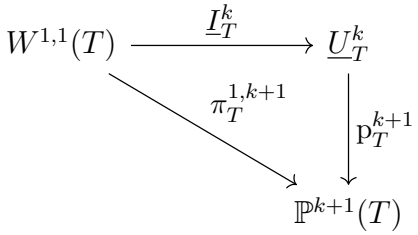


Figure 1.4: Illustration of the composition of $\underline{I}_T^k$ and p_T^{k+1} (1.23), from [2].

Local Contribution

Let us now arrange the final local contribution, approximating the continuous bilinear form (1.15) on each element of the mesh T with the discrete bilinear form $\mathrm{a}_T : \underline{U}_T^k \times \underline{U}_T^k \to \mathbb{R}$ such that, for all $\underline{u}_T, \underline{v}_T \in \underline{U}_T^k$,

$$\mathrm{a}_T(\underline{u}_T, \underline{v}_T) \coloneqq (\boldsymbol{\nabla} \mathrm{p}_T^{k+1} \underline{u}_T, \boldsymbol{\nabla} \mathrm{p}_T^{k+1} \underline{v}_T)_T + \mathrm{s}_T(\underline{u}_T, \underline{v}_T). \tag{1.24}$$

The first term is the standard Galerkin contribution, responsible for consistency - when $\underline{u}_T$ and $\underline{v}_T$ are the interpolates of polynomials we retrieve the original form a -, while the second one is a stabilisation bilinear form $\mathrm{s}_T : \underline{U}_T^k \times \underline{U}_T^k \to \mathbb{R}$, whose role is to ensure coercivity for the discrete problem (as it is the continuous one), and which is conceived in such a way that:

(S1) s_T is *symmetric and positive semidefinite*, since we wish a_T to be symmetric and positive semidefinite as it is a at the continuous level;

(S2) s_T ensures *stability and boundedness* of the local discrete bilinear form: $\exists\, \eta > 0$ independent of h and T such that, for all $\underline{v}_T \in \underline{U}_T^k$,

$$\eta^{-1} \|\underline{v}_T\|_{1,T}^2 \leq \mathrm{a}_T(\underline{v}_T, \underline{v}_T) \leq \eta \|\underline{v}_T\|_{1,T}^2 \tag{1.25}$$

(notice that $\|\cdot\|_{1,T}$ and the seminorm induced by a_T are equivalent);

(S3) s_T is *polynomial consistent*:

$$\mathrm{s}_T(\underline{I}_T^k w, \underline{v}_T) = 0 \qquad \forall w \in \mathbb{P}^{k+1}(T) \quad \text{and} \quad \forall \underline{v}_T \in \underline{U}_T^k. \tag{1.26}$$

$(S2)$ ensures the coercivity of the bilinear form while $(S3)$ its consistency with respect to polynomials of degree $k+1$. Moreover the requirement $(S3)$ suggests that s_T can be obtained penalising residuals that vanish for interpolates of polynomial functions in $\mathbb{P}^{k+1}(T)$. Paradigmatic examples of such residuals are provided by the *difference operators*:

$$\delta_T^k : \underline{U}_T^k \to \mathbb{P}^k(T); \qquad \delta_T^k \underline{v}_T \coloneqq \pi_T^{0,k}(\mathrm{p}_T^{k+1} \underline{v}_T - v_T), \tag{1.27}$$

$$\delta_{TF}^k : \underline{U}_T^k \to \mathbb{P}^k(F); \qquad \delta_{TF}^k \underline{v}_T \coloneqq \pi_F^{0,k}(\mathrm{p}_T^{k+1} \underline{v}_T - v_F) \quad \forall F \in \mathcal{F}_T. \tag{1.28}$$

The difference operators δ_T^k and δ_{TF}^k vanish for polynomial functions in $\mathbb{P}^{k+1}(T)$:

Proposition 1.3 (Polynomial consistency of the difference operators)**.** *It holds, for all*

$T \in \mathcal{T}_h$ and all $w \in \mathbb{P}^{k+1}(T)$,

$$\delta_T^k \underline{I}_T^k w = 0 \quad \text{and} \quad \delta_{TF}^k \underline{I}_T^k w = 0 \quad \forall F \in \mathcal{F}_T. \tag{1.29}$$

Proof. Let us observe that

$$\delta_T^k \underline{I}_T^k w = \pi_T^{0,k}(\mathrm{p}_T^{k+1} \underline{I}_T^k w - \pi_T^{0,k} w) = \pi_T^{0,k}(\pi_T^{1,k+1} w - w) = \pi_T^{0,k}(w - w) = 0,$$

where the first equality is the definition of $\delta_T^k \underline{I}_T^k$, the second one holds due to the composition of interpolant and projector operators (1.23) and the polynomial invariance (1.7) for $\pi_T^{0,k}$, and the third one again by polynomial invariance for $\pi_T^{1,k+1}$. □

Remark. Recalling the definition (1.20) of the local interpolator, one can check that the difference operators are actually obtained in a natural way:

$$(\delta_T^k \underline{v}_T, (\delta_{TF}^k \underline{v}_T)_{F \in \mathcal{F}_T}) = \underline{I}_T^k \mathrm{p}_T^{k+1} \underline{v}_T - \underline{v}_T. \tag{1.30}$$

Two examples of stabilisation bilinear forms which satisfy (S1)-(S3) are the following:
Example (Original HHO stabilisation) The original HHO stabilisation is obtained setting

$$\mathrm{s}_T(\underline{u}_T, \underline{v}_T) \coloneqq \sum_{F \in \mathcal{F}_T} h_F^{-1}((\delta_{TF}^k - \delta_T^k)\underline{u}_T, (\delta_{TF}^k - \delta_T^k)\underline{v}_T)_F. \tag{1.31}$$

Example (A stabilisation inspired by Virtual Elements) An expression for the stabilisation term inspired by the Virtual Elements is obtained setting

$$\mathrm{s}_T(\underline{u}_T, \underline{v}_T) \coloneqq h_T^{-2}(\delta_T^k \underline{u}_T, \delta_T^k \underline{v}_T)_T + \sum_{F \in \mathcal{F}_T} h_F^{-1}(\delta_{TF}^k \underline{u}_T, \delta_{TF}^k \underline{v}_T)_F. \tag{1.32}$$

Unlike in (1.31), both volumetric and boundary contributions are present. The negative powers of the element and face diameters in each term are again selected so as to ensure dimensional homogeneity with the consistency term.

Although the difference operators are not the only ones that vanish with interpolates of polynomials in $\mathbb{P}^{k+1}(T)$, it can be proved that s_T satisfies $(S3)$ if and only if s_T depends on its arguments only through the operators δ_T^k and δ_{TF}^k.

Lemma 1.1 (Dependency of s_T)**.** *Let $T \in \mathcal{T}_h$ and let $\mathrm{s}_T : \underline{U}_T^k \times \underline{U}_T^k \to \mathbb{R}$ be a symmetric bilinear form. Then, s_T satisfies the polynomial consistency (S3) if and only if it depends on its arguments only via the difference operators* (1.27)*.*

1.2.2. Discrete problem

Once we have set up the local contributions we can assemble the final discrete problem. Let us introduce the global HHO space of discrete unknowns:

$$\underline{U}_h^k \coloneqq \Big\{\underline{v}_h = ((v_T)_{T\in\mathcal{T}_h}, (v_F)_{F\in\mathcal{F}_h}) \ : \ v_T \in \mathbb{P}^k(T) \quad \forall T \in \mathcal{T}_h \text{ and } v_F \in \mathbb{P}^k(F) \quad \forall F \in \mathcal{F}_h\Big\}, \tag{1.33}$$

equipped with the global seminorm $\|\cdot\|_{1,h}$ such that, for all $\underline{v}_h \in \underline{U}_h^k$,

$$\|\underline{v}_h\|_{1,h} \coloneqq \left(\sum_{T\in\mathcal{T}_h} \|\underline{v}_T\|_{1,T}^2\right)^{\frac{1}{2}}, \tag{1.34}$$

where the local seminorm $\|\cdot\|_{1,T}$ was defined in (1.19).
If $\underline{v}_h \in \underline{U}_h^k$, $v_h \in \mathbb{P}^k(\mathcal{T}_h)$ is a broken polynomial such that

$$(v_h)_{|T} \coloneqq v_T \qquad \forall T \in \mathcal{T}_h. \tag{1.35}$$

Given a smooth function $v \in W^{1,1}(\Omega)$, the *global interpolator* $\underline{I}_h^k$ gives the global discrete unknowns:

$$\underline{I}_h^k : W^{1,1}(\Omega) \to \underline{U}_h^k, \qquad \underline{I}_h^k v \coloneqq ((\pi_T^{0,k} v)_{T\in\mathcal{T}_h}, (\pi_F^{0,k} v)_{F\in\mathcal{F}_h}). \tag{1.36}$$

The following subspace accounts for Dirichlet boundary conditions:

$$\underline{U}_{h,0}^k \coloneqq \left\{\underline{v}_h \in \underline{U}_h^k \ : \ v_F = 0 \quad \forall F \in \mathcal{F}_h^{\mathrm{b}}\right\}. \tag{1.37}$$

On $\underline{U}_{h,0}^k$ the global seminorm $\|\cdot\|_{1,h}$ becomes a norm thanks to the Poincaré inequality.

Lemma 1.2 (Discrete Poincaré inequality)**.** *There exists $C_{\mathrm{P}} > 0$ depending only on Ω, d, and ϱ such that, for all $\underline{v}_h \in \underline{U}_{h,0}^k$,*

$$\|v_h\| \le C_{\mathrm{P}} \|\underline{v}_h\|_{1,h}. \tag{1.38}$$

In order to check that the seminorm $\|\cdot\|_{1,h}$ is a norm on $\underline{U}_{h,0}^k$ we show that, for all $\underline{v}_h \in \underline{U}_{h,0}^k$, $\|\underline{v}_h\|_{1,h} = 0$ implies $\underline{v}_h = \underline{0}$. Let $\underline{v}_h \in \underline{U}_{h,0}^k$ be such that $\|\underline{v}_h\|_{1,h} = 0$. By the Poincaré inequality (1.38), we have $\|v_h\| = 0$, hence $v_T = 0$ for all $T \in \mathcal{T}_h$. Then, from the definition (1.19) of the norm $\|\cdot\|_{1,T}$, we also have that $\|v_F - v_T\|_F = 0$, hence $v_F = v_T = 0$ on F, for all $T \in \mathcal{T}_h$ and all $F \in \mathcal{F}_T$. Since any mesh face belongs to a set of faces $\mathcal{F}_T$ for at least

one mesh element $T \in \mathcal{T}_h$, this concludes the proof.

We can now define the global bilinear forms $\mathrm{a}_h : \underline{U}_h^k \times \underline{U}_h^k \to \mathbb{R}$ and $\mathrm{s}_h : \underline{U}_h^k \times \underline{U}_h^k \to \mathbb{R}$: For all $\underline{u}_h, \underline{v}_h \in \underline{U}_h^k$,

$$\mathrm{a}_h(\underline{u}_h, \underline{v}_h) \coloneqq \sum_{T\in\mathcal{T}_h} \mathrm{a}_T(\underline{u}_T, \underline{v}_T), \qquad \mathrm{s}_h(\underline{u}_h, \underline{v}_h) \coloneqq \sum_{T\in\mathcal{T}_h} \mathrm{s}_T(\underline{u}_T, \underline{v}_T) \tag{1.39}$$

and the stabilisation seminorm $|\cdot|_{\mathrm{s},h}$ such that, for all $\underline{v}_h \in \underline{U}_h^k$,

$$|\underline{v}_h|_{\mathrm{s},h} \coloneqq \mathrm{s}_h(\underline{v}_h, \underline{v}_h)^{\frac{1}{2}}. \tag{1.40}$$

Lemma 1.3 (Properties of a_h). *The bilinear form a_h enjoys the following properties:*

(i) Stability and boundedness. *For all $\underline{v}_h \in \underline{U}_{h,0}^k$, it holds with η as in (1.25) that*

$$\eta^{-1}\|\underline{v}_h\|_{1,h}^2 \le \|\underline{v}_h\|_{\mathrm{a},h}^2 \le \eta\|\underline{v}_h\|_{1,h}^2 \text{ with } \|\underline{v}_h\|_{\mathrm{a},h} \coloneqq \mathrm{a}_h(\underline{v}_h, \underline{v}_h)^{\frac{1}{2}}. \tag{1.41}$$

(ii) Consistency. *It holds for all $r \in \{0, \dots, k\}$ and all $w \in H_0^1(\Omega) \cap H^{r+2}(\mathcal{T}_h)$ such that $\Delta w \in L^2(\Omega)$,*

$$\sup_{\underline{v}_h \in \underline{U}_{h,0}^k, \|\underline{v}_h\|_{\mathrm{a},h}=1} |\mathcal{E}_h(w; \underline{v}_h)| \lesssim h^{r+1}|w|_{H^{r+2}(\mathcal{T}_h)}, \tag{1.42}$$

where the hidden constant is independent of w and h, and the linear form $\mathcal{E}_h(w;\cdot) : \underline{U}_{h,0}^k \to \mathbb{R}$ representing the consistency error is such that, for all $\underline{v}_h \in \underline{U}_{h,0}^k$,

$$\mathcal{E}_h(w; \underline{v}_h) \coloneqq -(\Delta w, v_h) - \mathrm{a}_h(\underline{I}_h^k w, \underline{v}_h). \tag{1.43}$$

Finally, the HHO discrete approximation of the Poisson problem (1.14) reads: Find $\underline{u}_h \in \underline{U}_{h,0}^k$ such that

$$\mathrm{a}_h(\underline{u}_h, \underline{v}_h) = (f, v_h) \qquad \forall \underline{v}_h \in \underline{U}_{h,0}^k. \tag{1.44}$$

Proving the well-posedness of (1.44) requires:

Lemma 1.4 (Lax–Milgram). *Let $\mathtt{U}$ be a real Hilbert space, let $\mathtt{a} : \mathtt{U} \times \mathtt{U} \to \mathbb{R}$ denote a bounded bilinear form, and let $\mathtt{f} \in \mathtt{U}^\star$, with $\mathtt{U}^\star$ denoting the dual space of $\mathtt{U}$. Further assume that the bilinear form $\mathtt{a}$ is $\mathtt{U}$-coercive, i.e., there exists a real number $C > 0$ such*

that, for all $v \in \mathtt{U}$,

$$C\|v\|_{\mathtt{U}}^2 \leq \mathtt{a}(v,v),$$

where $\|\cdot\|_{\mathtt{U}}$ *denotes the norm induced by the inner product in* $\mathtt{U}$. *Then, the problem: Find* $u \in \mathtt{U}$ *such that*

$$\mathtt{a}(u,v) = \langle \mathtt{f}, v\rangle_{\mathtt{U}^\star,\mathtt{U}} \qquad \forall v \in \mathtt{U},$$

is well-posed, i.e., it admits a unique solution for which the following a priori bound holds:

$$\|u\|_{\mathtt{U}} \leq C^{-1}\|\mathtt{f}\|_{\mathtt{U}^\star}.$$

Lemma 1.5 (Well-posedness of problem (1.44)). *Problem* (1.44) *is well-posed, and we have the following a priori bound for the unique discrete solution* $\underline{u}_h \in \underline{U}_{h,0}^k$*:*

$$\|\underline{u}_h\|_{\mathrm{a},h} \leq \eta^{\frac{1}{2}} C_{\mathrm{P}} \|f\|, \tag{1.45}$$

where C_{P} *denotes the constant of the discrete Poincaré inequality* (1.38) *and* η *is as in* (1.25).

Proof. Let us check the assumptions of the Lax–Milgram Lemma with $\mathtt{U} = \underline{U}_{h,0}^k$, $\mathtt{a} = \mathrm{a}_h$, and $\langle \mathtt{f}, \underline{v}_h\rangle_{\mathtt{U}^\star,\mathtt{U}} = (f, v_h)$. $\underline{U}_{h,0}^k$ equipped with the norm $\|\cdot\|_{\mathrm{a},h}$ is a Hilbert space. The bilinear form a_h is obviously coercive with respect to the norm $\|\cdot\|_{\mathrm{a},h}$ with coercivity constant equal to 1. Finally, by the discrete Poincaré inequality (1.38) and the norms equivalence (1.41), it holds that

$$|(f, v_h)| \leq \|f\|\ \|v_h\| \leq C_{\mathrm{P}}\|f\|\ \|\underline{v}_h\|_{1,h} \leq \eta^{\frac{1}{2}} C_{\mathrm{P}}\|f\|\ \|\underline{v}_h\|_{\mathrm{a},h}.$$

In particular this implies that the linear form $\mathtt{f} : \underline{v}_h \mapsto (f, v_h)$ is continuous □

- METTERE FLUX FORMULATION ? -

1.2.3. A priori error analysis

Being the problem well-posed, let us now study if the discrete solution is convergent to the exact one. We will see this result in two norms, the energy norm and the L^2 norm. Note that we cannot compare directly u and $\underline{u}_h$ since they live in different spaces, hence we will see the convergence result in two ways: comparing u to the potential reconstruction of $\underline{u}_u$, and comparing $\underline{u}_h$ to the interpolate of u.

Energy error estimate

We start with a convergence result in the discrete energy norm. Thanks to the Strang Lemma, we know that from a stability result follows an error estimate.

Theorem 1.2 (Discrete energy error estimate). *Let $(\mathcal{M}_h)_{h\in\mathcal{H}}$ denote a regular mesh sequence in the sense of Definition 1.5. Let a polynomial degree $k \geq 0$ be fixed. Let $u \in H^1_0(\Omega)$ denote the unique solution to (1.14), for which we assume the additional regularity $u \in H^{r+2}(\mathcal{T}_h)$ for some $r \in \{0,\dots,k\}$. For all $h \in \mathcal{H}$, let $\underline{u}_h \in \underline{U}^k_{h,0}$ denote the unique solution to (1.44) with stabilisation bilinear form s_T, $T \in \mathcal{T}_h$, in (1.24) satisfying Assumptions (S1)-(S3). Then,*

$$\|\underline{u}_h - \underline{I}^k_h u\|_{\mathrm{a},h} \lesssim h^{r+1}|u|_{H^{r+2}(\mathcal{T}_h)}, \tag{1.46}$$

where $\|\cdot\|_{\mathrm{a},h}$ is defined in (1.41) and the hidden constant is independent of h and u.

Proof. We use the Strang Lemma with $\mathtt{U} = H^1_0(\Omega)$, $\mathtt{a}(u,v) = (\boldsymbol{\nabla} u, \boldsymbol{\nabla} v)$, $\mathtt{l}(v) = (f,v)$, $\mathtt{U}_h = \underline{U}^k_{h,0}$ with norm $\|\cdot\|_{\mathrm{a},h}$, $\mathtt{a}_h = \mathrm{a}_h$, $\mathtt{l}_h(\underline{v}_h) = (f, v_h)$, and $\mathtt{I}_h u = \underline{I}^k_h u$. It follows that

$$\|\underline{u}_h - \underline{I}^k_h u\|_{\mathrm{a},h} \leq \gamma^{-1}\|\mathcal{E}_h(u;\cdot)\|_{\mathrm{a},h,\star}, \tag{1.47}$$

being $\mathcal{E}_h(u;\cdot)$ the consistency error (1.43) with $w = u$, as $-\Delta u = f$.

Moreover we know that the bilinear form $\mathtt{a}_h$ enjoys the consistence property (1.42), that we recall here for convenience:

$$\sup_{\underline{v}_h \in \underline{U}^k_{h,0}, \|\underline{v}_h\|_{\mathrm{a},h}=1} |\mathcal{E}_h(w;\underline{v}_h)| \lesssim h^{r+1}|w|_{H^{r+2}(\mathcal{T}_h)}, \tag{1.48}$$

Since we can recognise in the left-hand-side the dual energy norm of the consistency error, (1.46) follows plugging (1.48) into (1.47). □

This result gives us an estimate on the discrete error, defined as the difference between the discrete solution $\underline{u}_h$ and the interpolate of the continuous solution.

Let us now proceed showing a different error estimate, obtained comparing the continuous solution to the potential reconstruction of the discrete one. To this end we define a global potential reconstruction operator obtained piecewise from the local one: $\mathrm{p}^{k+1}_h : \underline{U}^k_h \to \mathbb{P}^{k+1}(\mathcal{T}_h)$ such that, for all $\underline{v}_h \in \underline{U}^k_h$,

$$(\mathrm{p}^{k+1}_h \underline{v}_h)_{|T} \coloneqq \mathrm{p}^{k+1}_T \underline{v}_T \qquad \forall T \in \mathcal{T}_h. \tag{1.49}$$

This operator allows now to compare directly $\mathrm{p}_h^{k+1}\underline{u}_h$ to u.

Theorem 1.3 (Energy error estimate for the reconstructed approximate solution)**.** *Under the assumptions and notations of Theorem 1.2, it holds that*

$$\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h - u)\| + |\underline{u}_h|_{\mathrm{s},h} \lesssim h^{r+1}|u|_{H^{r+2}(\mathcal{T}_h)}, \tag{1.50}$$

where the hidden constant is independent of h and u, and the $|\cdot|_{\mathrm{s},h}$ seminorm is defined by (1.40)*.*

We can notice that in the left-hand-side there are two terms, the first one linking the broken gradient of the global reconstruction to the gradient of u, and the second one related to the stabilisation form. The latter can be associated to the jumps of u, telling us that the jumps of u converge to 0 with h^{r+1}.

L^2-error estimate

Finally we state a result regarding the convergence of the error in L^2-norm.

Remark (Approximation properties of the discrete space) Notice that, because of the definition of global interpolator, thanks to the equivalence of the norms $\|\cdot\|_{1,h}$ and $\|\cdot\|_{1,a}$, and exploiting the previous error estimate in energy norm, it can be proved the following:

$$\|u_h - \pi_h^{0,k}u\| \lesssim h^{r+1}|u|_{H^{r+2}(\mathcal{T}_h)}, \tag{1.51}$$

where u_h is the discontinuous polynomial function given by the components of $\underline{u}_h$ corresponding to the each element T.

The key point now is to achieve an error estimate of higher degree, valid for HHO unknowns, which is called result of *superconvergence*, as the numerical solution converges to the projection of the exact one faster than the projection to the exact solution. This is possible adding further regularity to the continuous problem.

We, therefore, assume that for all $g \in L^2(\Omega)$, the unique solution of the dual problem: Find $z_g \in H_0^1(\Omega)$ such that

$$a(v, z_g) = (g, v) \qquad \forall v \in H_0^1(\Omega) \tag{1.52}$$

satisfies the *elliptic regularity*

$$\|z_g\|_{H^2(\Omega)} \leq C\|g\|, \tag{1.53}$$

with real number C depending only on Ω. Elliptic regularity holds when the domain Ω is convex.

To finally formalize the above idea let us state the following theorem.

Theorem 1.4 (Superconvergence of element unknowns)**.** *Let $(\mathcal{M}_h)_{h\in\mathcal{H}}$ denote a regular mesh sequence in the sense of Definition 1.5. Let a polynomial degree $k \geq 0$ be fixed. Let $u \in H_0^1(\Omega)$ denote the unique solution of (1.14), for which we assume the additional regularity $u \in H^{r+2}(\mathcal{T}_h)$ for some $r \in \{0,\dots,k\}$. For all $h \in \mathcal{H}$, let $\underline{u}_h \in \underline{U}_{h,0}^k$ denote the unique solution to (1.44) with stabilisation bilinear forms s_T, $T \in \mathcal{T}_h$, in (1.24) satisfying assumptions (S1)-(S3). Further assuming elliptic regularity and that $f \in H^1(\mathcal{T}_h)$ if $k = 0$, it holds that*

$$\|u_h - \pi_h^{0,k} u\| \lesssim \begin{cases} h^2 \|f\|_{H^1(\mathcal{T}_h)} & \text{if } k = 0, \\ h^{r+2} |u|_{H^{r+2}(\mathcal{T}_h)} & \text{if } k \geq 1, \end{cases} \tag{1.54}$$

where the hidden constant is independent of both h and u, and the global L^2-orthogonal projection $\pi_h^{0,k} u$ is defined according to (1.9), i.e., $(\pi_h^{0,k} u)_{|T} = \pi_T^{0,k} u_{|T}$ for all $T \in \mathcal{T}_h$.

As anticipated, adding the hypothesis of elliptic regularity to the continuous problem we gain an order of convergence with respect to the estimate 1.51, namely, the L^2-norm of the error converges as h^{k+2}. This means that element-based discrete unknowns superconverge to the L^2-orthogonal projection of degree k of the exact solution, provided sufficient regularity of the solution. This is one of the distinctive features of HHO methods.

1.3. Implementation

Let us start fixing a basis for $\mathbb{P}^l(T)$ for any mesh element $T \in \mathcal{T}_h$ and any integer $l \geq 0$, denoted by

$$\Phi_T^l \coloneqq \{\varphi_i^T\}_{1\leq i\leq N_d^l},$$

and a basis for $\mathbb{P}^k(F)$, for any face $F \in \mathcal{F}_h$, denoted by

$$\Phi_F^k \coloneqq \{\varphi_i^F\}_{1\leq i\leq N_{d-1}^k}.$$

Every function $v_T \in \mathbb{P}^l(T)$ and $v_F \in \mathbb{P}^k(F)$ can therefore be written as

$$v_T = \sum_{i=1}^{N_d^l} V_i^T \varphi_i^T \text{ for all } T \in \mathcal{T}_h \text{ and } v_F = \sum_{i=1}^{N_{d-1}^k} V_i^F \varphi_i^F \text{ for all } F \in \mathcal{F}_h, \tag{1.55}$$

where the real numbers V_i^T, V_i^F are called *degrees of freedom*.
The resulting basis for the global HHO space $\underline{U}_h^k$ is obtained taking the Cartesian product of the bases for the local polynomial spaces:

$$\Phi_h^k \coloneqq \left(\bigtimes_{T\in\mathcal{T}_h} \Phi_T^k\right) \times \left(\bigtimes_{F\in\mathcal{F}_h} \Phi_F^k\right).$$

1.4. Local construction and discrete problem

Let us now introduce briefly the algebraic formulation of the method for the Poisson problem starting from the local contributions. Let $\underline{v}_T \in \underline{U}_T^k$ be given, and denote by $\underline{\mathsf{V}}_T$ the corresponding vector of degrees of freedom partitioned as follows:

$$\underline{\mathsf{V}}_T = \begin{bmatrix} \mathsf{V}_T \\ \mathsf{V}_{F_1} \\ \vdots \\ \mathsf{V}_{F_{N_{\partial,T}}} \end{bmatrix} \in \mathbb{R}^{N_{\mathrm{dof},T}}$$

with subvectors

$$\mathsf{V}_T = [V_i^T]_{1\le i\le N_d^k} \in \mathbb{R}^{N_d^k}, \qquad \mathsf{V}_F = [V_i^F]_{1\le i\le N_{d-1}^k} \in \mathbb{R}^{N_{d-1}^k} \quad \forall F \in \mathcal{F}_T,$$

where, setting $N_{\partial,T} \coloneqq \mathrm{card}(\mathcal{F}_T)$, we have defined the integer

$$N_{\mathrm{dof},T} \coloneqq \dim(\underline{U}_T^k) = N_d^k + N_{\partial,T} N_{d-1}^k$$

representing the number of local degrees of freedom associated with T and its faces, and we have introduced a numbering of the faces of T from 1 to $N_{\partial,T}$.

Let us now recall the local bilinear form:

$$\mathrm{a}_T(\underline{u}_T, \underline{v}_T) \coloneqq (\boldsymbol{\nabla}\mathrm{p}_T^{k+1}\underline{u}_T, \boldsymbol{\nabla}\mathrm{p}_T^{k+1}\underline{v}_T)_T + \mathrm{s}_T(\underline{u}_T, \underline{v}_T).$$

This form is associated with a symmetric positive semidefinite local matrix $\mathbf{A}_T$, which represents the contribution of element T to the system matrix (see the definition (1.39) of $\mathbf{A}_h$).
This local contribution can be decomposed into its consistency and stability terms as

$$\mathbf{A}_T = \mathbf{A}_T^{\mathrm{cons}} + \mathbf{A}_T^{\mathrm{stab}} \in \mathbb{R}^{N_{\mathrm{dof},T}\times N_{\mathrm{dof},T}}. \tag{1.56}$$

The consistency contribution reads

$$\mathbf{A}_T^{\mathrm{cons}} = \mathbf{P}_T^{\intercal}\mathbf{S}_T\mathbf{P}_T,$$

where $\mathbf{P}_T$ is a matrix of size $N_d^{k+1} \times N_{\mathrm{dof},T}$ which represents the linear operator $\mathrm{p}_T^{k+1} : \underline{U}_T^k \to \mathbb{P}^{k+1}(T)$, once we have fixed a basis for $\underline{U}_T^k$ and one for $\mathbb{P}^{k+1}(T)$, and $\mathbf{S}_T \coloneqq \left[(\boldsymbol{\nabla}\varphi_i^T, \boldsymbol{\nabla}\varphi_j^T)_T\right]_{1\le i,j\le N_d^{k+1}}$.
The stability terms can be of different forms. The one corresponding to Example 1.31 is :

$$\mathbf{A}_T^{\mathrm{stab}} = \sum_{F\in\mathcal{F}_T} h_F^{-1}(\mathbf{D}_{TF} - (\mathbf{M}_{FF}^{k,k})^{-1}\mathbf{M}_{FT}^{k,k}\mathbf{D}_T)^{\intercal}\mathbf{M}_{FF}^{k,k}(\mathbf{D}_{TF} - (\mathbf{M}_{FF}^{k,k})^{-1}\mathbf{M}_{FT}^{k,k}\mathbf{D}_T), \quad (1.57)$$

where

- for integers $l, m \ge 0$, we define the local element mass matrix,

$$\mathbf{M}_{TT}^{l,m} \coloneqq \left[(\varphi_i^T, \varphi_j^T)_T\right]_{1\le i\le N_d^l, 1\le j\le N_d^m};$$

- the face-element and face-face mass matrices, for a fixed face $F \in \mathcal{F}_T$ and for given integers $l, m \ge 0$, are

$$\mathbf{M}_{FT}^{l,m} \coloneqq \left[(\varphi_i^F, \varphi_j^T)_F\right]_{1\le i\le N_{d-1}^l, 1\le j\le N_d^m}, \quad \mathbf{M}_{FF}^{l,m} \coloneqq \left[(\varphi_i^F, \varphi_j^F)_F\right]_{1\le i\le N_{d-1}^l, 1\le j\le N_{d-1}^m};$$

- $\mathbf{D}_{TF} \in \mathbb{R}^{N_{d-1}^k \times N_{\mathrm{dof},T}}$ represent the difference operators $\delta_{TF}^k : \underline{U}_T^k \to \mathbb{P}^k(F)$ and is defined such that:

$$\mathbf{D}_{TF} \coloneqq (\mathbf{M}_{FF}^{k,k})^{-1}\mathbf{M}_{FT}^{k,k+1}\mathbf{P}_T - \begin{bmatrix}\mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}_{N_{d-1}^k} & \cdots & \mathbf{0}\end{bmatrix};$$

- $\mathbf{D}_T \in \mathbb{R}^{N_d^k \times N_{\mathrm{dof},T}}$ represents the element difference operator $\delta_T^k : \underline{U}_T^k \to \mathbb{P}^k(T)$ and it is defined such that

$$\mathbf{D}_T \coloneqq (\mathbf{M}_{TT}^{k,k})^{-1}\mathbf{M}_{TT}^{k,k+1}\mathbf{P}_T - \begin{bmatrix}\mathbf{I}_{N_d^k} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}.$$

The local contribution from the element T to the source term is

$$\mathsf{B}_T = \begin{bmatrix}\mathsf{B}_T^T \\ 0\end{bmatrix} \in \mathbb{R}^{N_{\mathrm{dof},T}}, \qquad \mathsf{B}_T^T \coloneqq \left[(f, \varphi_i^T)_T\right]_{1\le i\le N_d^k},$$

where the $\mathsf{0}$ block fills the rows corresponding to all face unknowns.

We can now assemble the discrete problem element-wise.

$$\widetilde{\mathbf{A}}_h = \sum_{T\in\mathcal{T}_h} \mathbf{A}_T \qquad \widetilde{\mathsf{B}}_h = \sum_{T\in\mathcal{T}_h} \mathsf{B}_T. \tag{1.58}$$

We assume the following ordering for the degrees of freedom: first those attached to mesh elements, then those attached to interfaces and Neumann boundary faces, finally those attached to Dirichlet boundary faces. Defining the set of non-Dirichlet faces

$$\mathcal{F}_h^{\not\mathrm{D}} \coloneqq \mathcal{F}_h \setminus \mathcal{F}_h^{\mathrm{D}} = \mathcal{F}_h^{\mathrm{i}} \cup \mathcal{F}_h^{\mathrm{N}}, \tag{1.59}$$

this ordering induces the following block structure on $\widetilde{\mathbf{A}}_h$ and $\widetilde{\mathsf{B}}_h$:

$$\widetilde{\mathbf{A}}_h = \begin{bmatrix} \mathbf{A}_{\mathcal{T}_h\mathcal{T}_h} & \mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\not\mathrm{D}}} & \mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\mathrm{D}}} \\ \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{T}_h} & \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{F}_h^{\not\mathrm{D}}} & \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{F}_h^{\mathrm{D}}} \\ \mathbf{A}_{\mathcal{F}_h^{\mathrm{D}}\mathcal{T}_h} & \mathbf{A}_{\mathcal{F}_h^{\mathrm{D}}\mathcal{F}_h^{\not\mathrm{D}}} & \mathbf{A}_{\mathcal{F}_h^{\mathrm{D}}\mathcal{F}_h^{\mathrm{D}}} \end{bmatrix}, \qquad \widetilde{\mathsf{B}}_h = \begin{bmatrix} \mathsf{B}_{\mathcal{T}_h} \\ \mathsf{0} \\ \mathsf{0} \end{bmatrix}.$$

We account for the non-homogeneous Neumann boundary condition defining the following vector:

$$\mathsf{B}_{h,\mathrm{N}} = \Big[\mathsf{B}_{F,\mathrm{N}}\Big]_{F\in\mathcal{F}_h^{\not\mathrm{D}}} \text{ with } \mathsf{B}_{F,\mathrm{N}} \coloneqq \begin{cases} 0 & \text{if } F\in\mathcal{F}_h^{\mathrm{i}}, \\ \Big[(g_{\mathrm{N}},\varphi_i^F)_F\Big]_{1\le i\le N_{d-1}^k} & \text{if } F\in\mathcal{F}_h^{\mathrm{N}}. \end{cases}$$

Denoting by

$$N_{\mathrm{dof},h} \coloneqq \mathrm{card}(\mathcal{T}_h)N_d^k + \mathrm{card}(\mathcal{F}_h^{\not\mathrm{D}})N_{d-1}^k$$

the global number of degrees of freedom, the algebraic realisation of the discrete Poisson problem with mixed boundary conditions reads:

Find $\underline{\mathsf{U}}_{h,0} = \begin{bmatrix} \mathsf{U}_{\mathcal{T}_h,0} \\ \mathsf{U}_{\mathcal{F}_h^{\not\mathrm{D}},0} \end{bmatrix} \in \mathbb{R}^{N_{\mathrm{dof},h}}$ such that

$$\begin{bmatrix} \mathbf{A}_{\mathcal{T}_h\mathcal{T}_h} & \mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\not\mathrm{D}}} \\ \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{T}_h} & \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{F}_h^{\not\mathrm{D}}} \end{bmatrix} \begin{bmatrix} \mathsf{U}_{\mathcal{T}_h,0} \\ \mathsf{U}_{\mathcal{F}_h^{\not\mathrm{D}},0} \end{bmatrix} = \begin{bmatrix} \mathsf{B}_{\mathcal{T}_h} \\ \mathsf{B}_{h,\mathrm{N}} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\mathrm{D}}} \\ \mathbf{A}_{\mathcal{F}_h^{\not\mathrm{D}}\mathcal{F}_h^{\mathrm{D}}} \end{bmatrix} \mathsf{U}_{\mathcal{F}_h^{\mathrm{D}}} =: \begin{bmatrix} \mathsf{C}_{\mathcal{T}_h} \\ \mathsf{C}_{\mathcal{F}_h^{\not\mathrm{D}}} \end{bmatrix}. \tag{1.60}$$

1.4.1. Static condensation

HHO method is characterized by degrees of freedom attached to both cells and faces, however, it is a common practice to account only for face unknowns assembling the global system. This technique is known as static condensation. Hence HHO becomes a skeletal method, being the unknowns of the solution referring only to face degrees of freedom.

Notice that the submatrix $\mathbf{A}_{\mathcal{T}_h\mathcal{T}_h}$ is block-diagonal (with each block corresponding to one mesh element) and symmetric positive definite, and is therefore inexpensive to invert. The block-diagonal structure is a consequence of the fact that, for a fixed mesh element $T \in \mathcal{T}_h$, the discrete unknown u_T attached to T interacts with the other discrete unknowns only through the face unknowns u_F, $F \in \mathcal{F}_T$.
As a consequence, a good idea seems to solve the linear system (1.60) in two steps:

(i) First, we express $\mathsf{U}_{\mathcal{T}_h,0}$ in terms of $\mathsf{C}_{\mathcal{T}_h}$ and $\mathsf{U}_{\mathcal{F}_h^{\not\partial},0}$, solving the following computationally cheap equation:

$$\mathsf{U}_{\mathcal{T}_h,0} = \mathbf{A}_{\mathcal{T}_h\mathcal{T}_h}^{-1}\left(\mathsf{C}_{\mathcal{T}_h} - \mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\not\partial}}\mathsf{U}_{\mathcal{F}_h^{\not\partial},0}\right). \tag{1.61a}$$

This step is the actual *static condensation* step;

(ii) Second (the main step), face-based coefficients in $\mathsf{U}_{\mathcal{F}_h^{\not\partial},0}$ are obtained solving the following global problem, involving only quantities attached to the mesh skeleton:

$$\underbrace{\left(\mathbf{A}_{\mathcal{F}_h^{\not\partial}\mathcal{F}_h^{\not\partial}} - \mathbf{A}_{\mathcal{F}_h^{\not\partial}\mathcal{T}_h}\mathbf{A}_{\mathcal{T}_h\mathcal{T}_h}^{-1}\mathbf{A}_{\mathcal{T}_h\mathcal{F}_h^{\not\partial}}\right)}_{=:\mathbf{A}_h^{\mathrm{sc}}}\mathsf{U}_{\mathcal{F}_h^{\not\partial},0} = \mathsf{C}_{\mathcal{F}_h^{\not\partial}} - \mathbf{A}_{\mathcal{F}_h^{\not\partial}\mathcal{T}_h}\mathbf{A}_{\mathcal{T}_h\mathcal{T}_h}^{-1}\mathsf{C}_{\mathcal{T}_h}. \tag{1.61b}$$

This step requires to invert the symmetric positive definite matrix $\mathbf{A}_h^{\mathrm{sc}}$, with size

$$N_{\mathrm{dof},h}^{\mathrm{sc}} := \mathrm{card}(\mathcal{F}_h^{\not\partial})N_{d-1}^k. \tag{1.61c}$$

2 | A posteriori error estimators

This chapter deals with a posteriori residual-based error estimators, whose relevance is related to mesh adaptation. A posteriori-driven mesh refinement gives some crucial advantages, such as the significant enhancement of the performance of problems with singular solutions, allowing to fully exploit the high-order of approximation of HHO schemes. Indeed, for smooth exact solutions, increasing the polynomial degree yields a corresponding increase in the convergence rate, as we saw in the previous section. However, if this is not the case and instead the exact solution is not regular enough, the order of convergence is limited by the poor regularity of the solution. In order to restore optimal order of convergence local mesh adaptation can help, typically using local a posteriori error estimators to mark the elements with the largest error, so that they will be refined. The analysis takes into consideration [2] as well as [1] and [4] for the residual based approach and the Fichera corner benchmark problem.

2.1. Reliable upper bound

The goal of this section is to prove an upper bound of the discretization error of the following form

$$\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h - u)\| \lesssim \varepsilon, \tag{2.1}$$

where the hidden constant is independent of the meshsize and of the problem data.

If ε, the *estimator*, is computable only through the discrete solution and the problem data, this upper bound is said to be *reliable*.

For polytopal meshes we will see an even stronger property than reliability: we will obtain an estimate of the form

$$\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h - u)\| \leq \varepsilon.$$

This time there are no undetermined constants in the right-hand side, so that the upper bound is said to be *guaranteed* and *fully computable*.

Let us now proceed briefly introducing some definitions used to derive error estimators

exploiting a residual-based approach .

For any integer $l \geq 1$ and any broken polynomial function $v_h \in \mathbb{P}^l(\mathcal{T}_h)$, let the residual $\mathcal{R}(v_h) \in H^{-1}(\Omega)$ be such that, for all $\varphi \in H_0^1(\Omega)$,

$$\langle \mathcal{R}(v_h), \varphi \rangle_{-1,1} \coloneqq a(u - v_h, \varphi) = (f, \varphi) - a(v_h, \varphi), \tag{2.2}$$

where $\langle \cdot, \cdot \rangle_{-1,1}$ denotes the duality pairing between $H^{-1}(\Omega)$ and $H_0^1(\Omega)$.

Let now the boundary residual operator $\underline{R}_{\partial T}^k : \underline{U}_T^k \to \underline{D}_{\partial T}^k$ be such that, for all $\underline{v}_T \in \underline{U}_T^k$, the vector of polynomials

$$\underline{R}_{\partial T}^k \underline{v}_T \coloneqq (R_{TF}^k \underline{v}_T)_{F \in \mathcal{F}_T}$$

satisfies, for all $\underline{\alpha}_{\partial T} = (\alpha_{TF})_{F \in \mathcal{F}_T} \in \underline{D}_{\partial T}^k$,

$$-\sum_{F \in \mathcal{F}_T} (R_{TF}^k \underline{v}_T, \alpha_{TF})_F = \mathrm{s}_T((0, \underline{\Delta}_{\partial T}^k \underline{v}_T), (0, \underline{\alpha}_{\partial T})). \tag{2.3}$$

The residual-based approach used to derive error estimators relies on the following abstract estimate.

Lemma 2.1 (Abstract estimate). *Let $u \in H_0^1(\Omega)$ solve the Poisson problem (1.14). Then, for any integer $l \geq 1$ and any broken polynomial function $v_h \in \mathbb{P}^l(\mathcal{T}_h)$, it holds that*

$$\|\boldsymbol{\nabla}_h(u - v_h)\|^2 \leq \inf_{\varphi \in H_0^1(\Omega)} \|\boldsymbol{\nabla}_h(\varphi - v_h)\|^2 + \left(\sup_{\varphi \in H_0^1(\Omega), \|\boldsymbol{\nabla}\varphi\| = 1} \langle \mathcal{R}(v_h), \varphi \rangle_{-1,1} \right)^2. \tag{2.4}$$

We can observe that the difference between the continuous solution of the Poisson problem, u, and a generic broken polynomial function v_h is estimated by two terms: a *nonconformity* term that measures the difference between v_h and the closest element of $H_0^1(\Omega)$ and a *residual* term measuring how far is v_h from being the exact solution of the Poisson problem in terms of the dual norm of the residual 2.2.

From this estimate it's possible to prove the upper bound we were looking for, which is the following.

Theorem 2.1 (A posteriori error upper bound). *Let $\mathcal{M}_h$ denote a polytopal mesh in the sense of Definition 1.2, and let an integer $k \geq 0$ be fixed. Let $u \in H_0^1(\Omega)$ and $\underline{u}_h \in \underline{U}_{h,0}^k$ denote the unique solutions to problems (1.14) and (1.44), respectively, with local stabilisation bilinear forms s_T, $T \in \mathcal{T}_h$, satisfying assumptions (S1)-(S3). Then, it*

holds that

$$\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h - u)\| \le \varepsilon \coloneqq \left[\sum_{T\in\mathcal{T}_h}\left(\varepsilon_{\mathrm{nc},T}^2 + (\varepsilon_{\mathrm{res},T} + \varepsilon_{\mathrm{sta},T})^2\right)\right]^{\frac{1}{2}}, \tag{2.5}$$

with local nonconformity*,* residual*, and* stabilisation *estimators such that, for all* $T \in \mathcal{T}_h$*,*

$$\varepsilon_{\mathrm{nc},T} \coloneqq \|\boldsymbol{\nabla}(\mathrm{p}_T^{k+1}\underline{u}_T - u_h^*)\|_T, \tag{2.6a}$$

$$\varepsilon_{\mathrm{res},T} \coloneqq C_{\mathrm{P},T}h_T\|(f + \Delta\mathrm{p}_T^{k+1}\underline{u}_T) - \pi_T^{0,0}(f + \Delta\mathrm{p}_T^{k+1}\underline{u}_T)\|_T, \tag{2.6b}$$

$$\varepsilon_{\mathrm{sta},T} \coloneqq C_{\mathrm{F},T}^{\frac{1}{2}}h_T^{\frac{1}{2}}\left(\sum_{F\in\mathcal{F}_T}\|R_{TF}^k\underline{u}_T\|_F^2\right)^{\frac{1}{2}}, \tag{2.6c}$$

where u_h^* *is an arbitrary function in* $H_0^1(\Omega)$ *and, for all* $F \in \mathcal{F}_T$*, the boundary residual operator* R_{TF}^k *is defined by* (2.3)*.*

In 2.5 you can still distinguish the different contributions coming from 2.4: $\sum_{T\in\mathcal{T}_h}\varepsilon_{\mathrm{nc},T}^2$ coming directly from the nonconformity term and the rest coming from the residual part.

Remark. The function u_h^* can be obtained from the HHO discrete solution $\underline{u}_h$ by applying the node-averaging operator, defined below, to the global potential reconstructor $p_h^{k+1}u_h$.

Let an integer $l \ge 1$ be fixed. When $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ is a matching simplicial mesh in the sense of Definition 1.3, the node-averaging operator $\mathcal{I}_{\mathrm{av},h}^l : \mathbb{P}^l(\mathcal{T}_h) \to \mathbb{P}^l(\mathcal{T}_h) \cap H_0^1(\Omega)$ is defined by setting, for each Lagrange interpolation node $\boldsymbol{N}$,

$$(\mathcal{I}_{\mathrm{av},h}^l v_h)(\boldsymbol{N}) \coloneqq \begin{cases} \dfrac{1}{\mathrm{card}(\mathcal{T}_{\boldsymbol{N}})}\displaystyle\sum_{T\in\mathcal{T}_{\boldsymbol{N}}}(v_h)_{|T}(\boldsymbol{N}) & \text{if } \boldsymbol{N}\in\Omega, \\ 0 & \text{if } \boldsymbol{N}\in\partial\Omega, \end{cases}$$

where the set $\mathcal{T}_{\boldsymbol{N}} \subset \mathcal{T}_h$ collects the simplices to which $\boldsymbol{N}$ belongs. We then set

$$u_h^* \coloneqq \mathcal{I}_{\mathrm{av},h}^{k+1}\mathrm{p}_h^{k+1}\underline{u}_h. \tag{2.7}$$

The generalisation to polytopal meshes can be realised applying the node averaging operator to $\mathrm{p}_h^{k+1}\underline{u}_h$ on a matching simplicial submesh of $\mathcal{T}_h$ (whose existence is guaranteed for regular mesh sequences, see Definition 1.5).

2.2. Local and global efficiency

In this section we will show that the previous estimate is both local and global efficient by showing that it is bounded from above by the discretisation error. This is important to be sure that the estimators localise the error correctly and do not overstimate it.

Let us start with the *local efficiency*. An error estimator on a given mesh element $T \in \mathcal{T}_h$ is local efficient if it provides a local lower bound, namely if it is bounded by the approximation error on some elements surrounding T. This implies that the a posteriori estimate is eligible to drive local mesh refinement.

The local efficiency of the previous estimators is given by the following theorem.

Theorem 2.2 (A posteriori local error lower bound)**.** *We let the assumptions of Theorem 2.1 hold and further assume, for the sake of simplicity, that*

(i) for all $T \in \mathcal{T}_h$, the stabilisation bilinear form s_T is given by (1.32);

(ii) the H_0^1-conforming reconstruction $u_h^\star$ is obtained using the node-averaging operator on the matching simplicial submesh $\mathfrak{M}_h = (\mathfrak{T}_h, \mathfrak{F}_h)$ of $\mathcal{M}_h = (\mathcal{T}_h, \mathcal{F}_h)$ of Definition 1.5;

(iii) we have, for the forcing term, $f \in \mathbb{P}^{k+1}(\mathcal{T}_h)$.

Then, it holds, for all $T \in \mathcal{T}_h$,

$$\varepsilon_{\mathrm{nc},T} \lesssim \left(\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h - u)\|_{\mathcal{N},T} + |\underline{u}_h|_{\mathrm{s},\mathcal{N},T} \right), \tag{2.8a}$$

$$\varepsilon_{\mathrm{res},T} \lesssim \|\boldsymbol{\nabla}(\mathrm{p}_T^{k+1}\underline{u}_T - u)\|_T, \tag{2.8b}$$

$$\varepsilon_{\mathrm{sta},T} \lesssim |\underline{u}_T|_{\mathrm{s},T}, \tag{2.8c}$$

with hidden constants possibly depending on d, ϱ, and on k, but independent of h, T, and u. For all $T \in \mathcal{T}_h$, $\|\cdot\|_{\mathcal{N},T}$ denotes the L^2-norm on the union of the elements in $\mathcal{T}_{\mathcal{N},T}$ and we have set, with stabilisation seminorm $|\cdot|_{\mathrm{s},T'}$, for $T' \in \mathcal{T}_{\mathcal{N},T}$, such that, for all $\underline{v}_{T'} \in \underline{U}_{T'}^k$, $|\underline{v}_{T'}|_{\mathrm{s},T'}^2 \coloneqq \mathrm{s}_{T'}(\underline{v}_{T'}, \underline{v}_{T'})$,

$$|\underline{u}_h|_{\mathrm{s},\mathcal{N},T} \coloneqq \left(\sum_{T' \in \mathcal{T}_{\mathcal{N},T}} |\underline{u}_{T'}|_{\mathrm{s},T'}^2 \right)^{\frac{1}{2}}.$$

As immediate consequence, the global lower bound holds. This entails *global efficiency* of the estimator:

Theorem 2.3 (Global lower bound)**.** *Under the assumptions of Theorem 2.2, it holds*

that

$$\left[\sum_{T\in\mathcal{T}_h}\left(\varepsilon_{\mathrm{nc},T}^2+(\varepsilon_{\mathrm{res},T}+\varepsilon_{\mathrm{sta},T})^2\right)\right]^{\frac{1}{2}}\lesssim\left(\|\boldsymbol{\nabla}_h(\mathrm{p}_h^{k+1}\underline{u}_h-u)\|+|\underline{u}_h|_{\mathrm{s},h}\right),$$

with hidden constant independent of h and f, but possibly depending on d, ϱ and k.

2.3. Numerical examples: a posteriori-driven mesh adaptivity

In this section we will see how to incorporate a posteriori estimators into mesh adaptivity, showing then some numerical examples carried out by [2] exploiting a conforming refinement of the mesh.

The typical adaptive procedure is characterized by the iteration of the following four steps:

- solve for the numerical scheme on the current mesh;
- compute the error estimator;
- mark certain elements with precise values of the estimators;
- refine them to get the next mesh.

Algorithm 2.1 Pseudocode of the automatic mesh adaptation procedure.

Set a tolerance tol > 0 and a maximum number of iterations $N_{\max}$
Generate an initial coarse mesh $\mathcal{T}_h^{(0)}$, set $n \leftarrow 0$, and let $\mathcal{T}_h^{(n)} \leftarrow \mathcal{T}_h^{(0)}$
repeat
  Solve the HHO problem (1.44) on $\mathcal{T}_h^{(n)}$
  for $T \in \mathcal{T}_h^{(n)}$ **do**
    Compute and store the local estimator $\varepsilon_T \coloneqq \left[\varepsilon_{\mathrm{nc},T}^2+(\varepsilon_{\mathrm{res},T}+\varepsilon_{\mathrm{sta},T})^2\right]^{\frac{1}{2}}$
  end for
  for $T \in \mathcal{T}_h^{(n)}$ **do**
    if T is among the 5% elements with the largest local estimator **then**
      Set a target diameter of $h_T/2$
    else
      Set a target diameter of h_T
    end if
  end for
  Set $n \leftarrow n+1$ and generate a novel mesh $\mathcal{T}_h^{(n)}$ by using the target element diameters
until $\varepsilon <$ tol **or** $n > N_{\max}$

The following numerical tests are performed on the Fichera corner benchmark problem, whose solution is known to be singular:

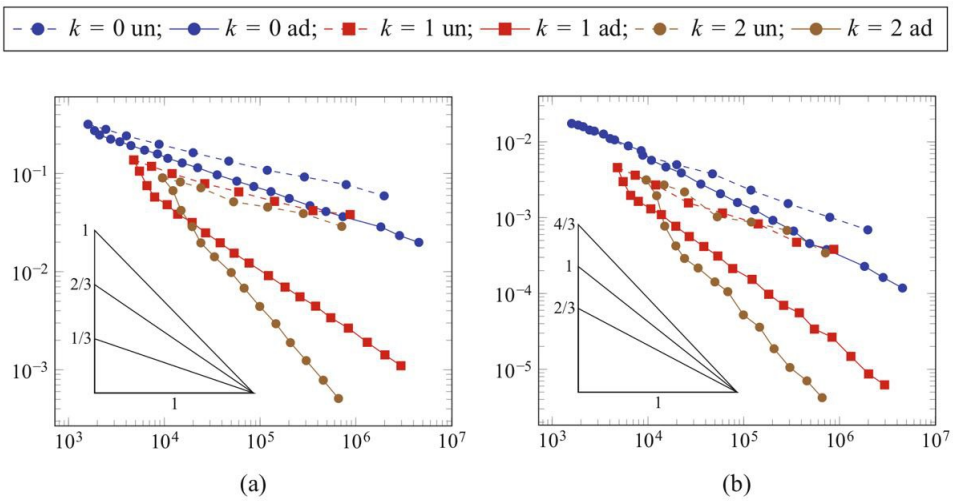


Figure 2.1: Error vs. N_{dof} for the test case of Fichera problem. "un"= uniformly refined meshes, "ad"= adaptively refined meshes. Figure taken from [2].

$$u(x_1, x_2, x_3) = \sqrt[4]{x_1^2 + x_2^2 + x_3^2}.$$

In Figure 2.1 it is plotted the numerical error versus the number of degrees of freedom N_{dof} on uniformly and adaptively refined mesh sequences for polynomial degrees up to 2.

Notice that the error is plotted against the number of degrees of freedom instead of the global meshsize h since h may not vary locally refining the mesh.

We can clearly see that with uniformly refined meshes the convergence order is not optimal, being limited by the poor regularity of the solution. On adaptively refined mesh sequences, on the other hand, the optimal convergence rates of $N_{dof}^{\frac{(k+1)}{d}}$ and $N_{dof}^{\frac{(k+2)}{d}}$ in both the energy-norm and L^2-norm are recovered. This confirms that Algorithm 2.1 is able to restore optimal orders of convergence.

3 | Non-conforming mesh adaptivity

An important asset of polytopal methods is the ability to treat non-conforming meshes. This is due to the fact that, by introducing a hanging node on an edge in the mesh, it can be seen as the union of other two different coplanar edges and the adjacent cell will be considered as a polytope with one more edge.
In this work, new non-conforming refinement methods for 2D meshes have been developed, and in this chapter they will be presented.

The theoretical benefit from a non-conforming refinement is being able to refine the mesh just locally, without adjusting and reconstructing the mesh in the entire domain, but only in a small area. This is crucial when you cope with phenomena which occur in one point or in a small part of the domain and it results in a significant computational cost reduction. The implementation rationale is, therefore, to make this theoretical concept a concrete advantage, by reproducing it at a computational level. As a consequence, the goal is to perform at each refinement only local operations.

Let us notice that the notion of non-conforming refinement of the mesh naturally leads to the notion *dynamic mesh*. Indeed, if we were interested in locally refining the mesh to better capture particular physical phenomena, we may want to coarsen it where the phenomena are less relevant or smoother. Therefore we will introduce a new `DynamicMesh` class, which has the simultaneous potential of both refining and coarsening the mesh.

3.1. HArD::Core library

Let us start by briefly introducing the HArD::Core library, the essential core for the following development of new methods and classes. HArD::Core provides a suite of C++ tools to implement numerical schemes on general polytopal meshes, whose unknowns are polynomials in the cells, both on the edges and on the faces (hybrid methods). The aim

is to deal with generic polytopal meshes. The library was mainly designed for the Hybrid High-Order methods: several HHO methods are already implemented in this framework, however the tools provided in this library are useful for a range of numerical methods.

The main sections of the library are the following modules:

- a set of classes and functions to represent suitable basis of polynomial functions in $\mathbb{R}^d$ with $d = 2, 3$;
- classes providing cell and edge quadrature rules, and values of basis functions at the nodes;
- HHO scheme classes, which load the mesh, properly impose boundary conditions, assemble the global system, and solve it;
- a set of classes to construct and describe polyhedral meshes and their entities. Suitable classes represent vertices, edges and cells (in the 2d case), and the mesh object can be seen as a collection of its enties, containing vectors of vertices, edges and cells. This is the module we will focus on.

3.1.1. The Mesh class

In order to handle the mesh, the main item is the `Mesh` class, which contains vectors of `Cell`, `Edge` and `Vertex` objects, based on classes that suitably represent all the entities of the mesh. The mesh class provides also methods to extract single elements, such as particular cells, edges, or vertices with a given index, as you can see below. For instance, the function `Mesh::get_cells()` returns a vector with all the cells of the mesh, ordered according to their global indeces, while the function `Mesh::cell(size_t i)` returns the cell with global index `i`.

Listing 3.1: The Mesh class with its main methods and attributes

```
class Mesh {

private:
    // primary data: list of cells, edges, vertices...
    std::vector<Cell*> _cells;
    std::vector<Edge*> _edges;
    std::vector<Vertex*> _vertices;
    std::vector<Cell*> _b_cells; // boundary cells
    std::vector<Edge*> _b_edges;
    ...
```

```
public:
    Mesh();
    ~Mesh();

    inline size_t n_cells() const;      // number of cells in the mesh
    inline size_t n_edges() const;      // number of edges in the mesh
    inline size_t n_vertices() const;   // number of vertices in the mesh
    size_t n_b_cells() const;           // number of boundary cells
    ...

    inline double h_max() const;        // max of diameter of cells

    inline std::vector<Cell*> get_cells() const; // lists  all the cells
    inline std::vector<Edge*> get_edges() const;  // lists the edges
    inline std::vector<Vertex*> get_vertices() const;  // lists the
       vertices
    Cell* cell(size_t iC) const;  // get a constant pointer to a cell
       using its global index
    Edge* edge(size_t iE) const;   // get a constant pointer to an edge
       using its global index
    Vertex* vertex(size_t iV) const;   // get a constant pointer to a
       vertex using its global index
    ...

};
```

3.2. DynamicMesh class

The goal of this work is to develop a new class which extends the classical `Mesh`, called `DynamicMesh`, which mantains the same interface as `Mesh`. In this way the two classes will be interchangeable and all the already existing schemes for solving partial differential equations, which use the `Mesh` class and all its methods, will not need to be changed.

Hence, the aim of our `DynamicMesh` class will be to add some new methods to refine the mesh, and in the meantime to overload the already existing getter functions `get_cells()`, `cell(size_t i)`, etc. Clearly they have to work in a more sophisticated way, assuming the mesh has been refined. Particularly, they will have to iterate over the mesh taking into account properly all the sub-elements coming from the refined elements.

Listing 3.2: Structure of the DynamicMesh class

```
class DynamicMesh: public Mesh {
```

```
public:
    //overloaded Mesh functions

    inline size_t n_cells() const;      // number of cells
    inline size_t n_edges() const;      // number of edges
    inline size_t n_vertices() const;   // number of vertices

    inline std::vector<Cell*> get_cells() const; // lists all the cells
    inline std::vector<Edge*> get_edges() const;  // lists all the edges
    inline std::vector<Vertex*> get_vertices() const;  // all vertices
    Cell* cell(size_t iC) const;  // pointer to the cell iC
    Edge* edge(size_t iE) const;   // pointer to the edge iE
    Vertex* vertex(size_t iV) const;   //  pointer to the vertex iV
    ...

    // some new methods and attributes to perform adaptivity of the mesh
    ... // see the following sections

};
```

Let us notice that, in this work, we will treat only the 2D case with triangular mesh elements that will be refined into four smaller cells joining the mid points of the original one, as shown in figure 3.1.

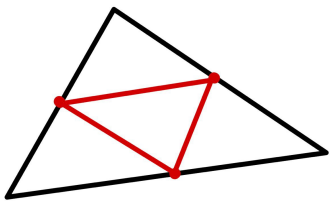


Figure 3.1: 2D triangular refined cell.

The basic mechanism is therefore the following: when a cell (edge) is refined, it becomes *father* of four (two) new *children* cells (edges, respectively).

Pay attention to the fact that the vectors of cells, edges and vertices already contained in the `DynamicMesh` object, inherited from the `Mesh` class, will not actually change. They are indeed private members of the `Mesh` class, and, from a coding point of view, they cannot be modified. On the other hand, even theoretically we do not want to modify them, neither we need it. Hence they will remain untouched. What will change is only the *information*

carried by the refinement operation, in such a way that our `DynamicMesh` object becomes a sort of virtual mesh, which contains all the information about the refinements but not every concrete element of the mesh.

How to store the information? To this purpose, some new ad-hoc classes have been created, namely `CellInfo` and `EdgeInfo`, which represent each cell and edge, with their global indices, and contain a vector of (pointers to their) children - possibly empty - and (a pointer to) their father, if any. They are stored in two vectors called `_cells_infos` and `_edges_infos` within the `DynamicMesh` class, in such a way that, at the beginning, when the mesh is still not refined, there is a correspondence 1 to 1 between each `CellInfo` (`EdgeInfo`) and each `Cell` (`Edge`).

Listing 3.3: CellInfo and EdgeInfo classes

```
class CellInfo{

private:
    size_t _Gi; // global index of the cell
    size_t _Li; // local index (among the cells in the same level of
       refinement)
    size_t _has_children = 0; // number of children and subchildren
    size_t _has_father = 0; // number of ancestors
    std::vector<CellInfo*> _children;
    CellInfo* _father = nullptr;

    size_t _n_vertices; // always 3
    std::vector<Vector2d> _coords; //vertices of the cell (always the
       original three vertices, even if the cell is refined)
    std::vector<CellInfo*> _neighbours; // the 3 "big" neighbours of the
        cell, possibly "virtual" cells, if they are actually refined
    std::vector<EdgeInfo*> _edges; //the 3 big edges, which can be "
       virtual" and have children
    std::vector<EdgeInfo*> _children_edges; // the 3 new edges that join
        the midpoints of the cell's edges when it is refined
    ...

public:
    std::vector<size_t> vertices_ids(); // it looks at the edges of the
       cell and takes consequently the indices of the vertices
    // getter functions
    ...
}
```

```
    class EdgeInfo{
```

```
        // same structure ad CellInfo

    }
```

3.2.1. The refinement function

At every cell refinement, given the global index of the cell to be refined, what happens is basically the following.

1. The corresponding (with same global index) `CellInfo` is located.
2. Four new `CellInfo` objects are created, namely the cell's children, with as `father` attribute the old cell.
3. The children attribute of the refined `CellInfo` will be filled with its new children.
4. Two new `EdgeInfo` objects per each edge of the cell are created, neamely the children of each edge, with the corresponding `father`.
5. The `children` attribute of each edge is filled accordingly.

In this way a tree-like structure will be generated, with every cell pointing to its children, and to its father. Notice moreover that every `CellInfo` contains also (the pointers to) its neighbours. This is necessary since the neighbours of a refined cell will change number of edges and vertices, so, ideally, the neighbours have to be identified and modified (Fig. 3.2). We will see later more in detail how this is carried out, looking at the function `DynamicMesh::cell(size_t i)`.

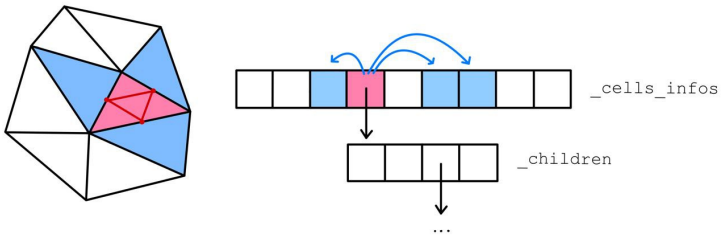


Figure 3.2: Tree-structure of a refined mesh.

Notice that the `CellInfo` attributes `vertices`, `edges` and `neighbours` contain always only three elements, even if the cell is refined and has actually more of them. We can think of them as *virtual* `edges` and `neighbours`, and, in order to catch the real ones, one needs to go deeply into their children and subchildren.

This is indeed the main point of the new class, namely being able to iterate correctly through all the elements and through all the children of each element. The overloading getter functions which return the elements of the mesh (like `DynamicMesh::get_cells()`, `DynamicMesh::cell(size_t i)` etc.) will have indeed to iterate through all the elements, checking if they have children, and if so, through all the children, in order to emulate the behaviour of the overloaded functions (`Mesh::get_cells()`, `Mesh::cell(size_t)`, etc.) and to return the right elements. In order to do it, recursive functions are exploited, which go deep into each element's children, until they reach the last level, meaning cells with no children anymore.

Listing 3.4: Overview of the refinement method

```
void DynamicMesh::refine(CellInfo* c) {

    _n_ref++; // number of refined cells in the mesh

    // - initialising the new elements

    std::vector<CellInfo*> new_cells(4); //The 4 new little cells
    std::vector<EdgeInfo*> central_edges(3); //The 3 new central edges
    ...

    // - refining the edges

    size_t new_vertex_idx = n_vertices();

    for (auto& e : c->_edges){ // cicle over the "big" edges of the cell
        if(!e->_has_children) {
            edge_refine(e, new_vertex_idx);
            new_vertex_idx++;
        }
        else e->_refined_cells++;
    }

    // - storing the vertices' coordinates and the vertices' indeces of
       the new elements
   ...

    // - creating the first 3 new cells and the central edges

    for (size_t i = 0; i < 3; i++){
        new_cells[i] = new CellInfo(find_global_index(c) + i, i,
           vertices[i]);
```

```
        ...
        central_edges[i] = new EdgeInfo(0, next_index_edge + i, verts,
            ids);
        central_edges[i]->_father_cell = c;
        c->_children_edges.push_back(central_edges[i]);
        ...
        new_cells[i]->_father = c;
        c->_children.push_back(new_cells[i]);
    }

    // - adjusting the indeces
    ...

    //  - creating the 4th new cell
    new_cells[3] = new CellInfo(find_global_index(c) + 3, 3, vertices
        [3]);
    new_cells[3]->_has_father = c->_has_father + 1;
    new_cells[3]->_father = c;
    c->_children.push_back(new_cells[3]);
    ...
```

Remark about indexing. Let us explain briefly how the indices of the elements are adjusted. Given a cell with index `iC` which we want to refine, its children will take indices `iC`, `iC+1`, `iC+2` and `iC+3`. The indices of all the subsequent elements must be shifted accordingly. The same happens for the edges. However, the new edges created inside the refined cell, joining the midpoints of its old edges, do not have a father and so they are added at the end, both in terms of indexing and in terms of physical vector; they are indeed added at the bottom of the `_edges_infos` vector. Similarly, regarding the new vertices, since they do not have a direct father, they are ideally added at the end - *ideally*, because notice that it does not exists a vector of vertices stored in the `DynamicMesh` class - and so they take as indices the last ones in the numbering. Finally let us remark that, while for sake of simplicity in this presentation has been shown the global index of the element `_iG_` in each `CellInfo` and `EdgeInfo`, actually, to avoid to iterate all over the elements every time a cell is refined in order to update the global index, our choice was to not consider it as attribute of the class. What is actually done is instead to add a public method `DynamicMesh::find_global_index(CellInfo* c)` that iterates over the elements and returns the right global index. This choice can be easily reversed according to which operations are performed the most in each circumstance.

Recalling now our initial goal, which was to try to perform just local operations, let

us explain briefly how it has been achieved.

Each time an element (a cell, or an edge, or a vertex) is asked to be returned by the getter functions of the mesh, the new overloading getter functions of `DynamicMesh` check if the element is still unchanged with respect to the original mesh, or it has changed due to the refinement operations. In the first case it returns (a pointer to) the original element already present in the `Mesh` vectors storing the original elements, while in the latter case it creates a new element from scratch.

Let us make a practical example. If the function `DynamicMesh::cell(iC)` is called, it must return the cell with iC-th global index. Hence, what it does is the following:

1. it locates the corresponding `CellInfo` (with global index `iC`);
2. it checks if that `CellInfo` has been refined (namely, it has a non-empty children vector) or if it has some refined neighbours;
3. if the answer is no, it returns (read: a pointer to) the original corresponding cell;
4. if yes, it creates a new `Cell` with the right number of edges and vertices, and returns it.

As a result, just the cells which are explicitly refined, and their neighbours, will be re-created, while all the others will be kept the same.

We conclude this section by showing one of the main bricks of all the implemented functions, namely the method `DynamicMesh::find_cell_info(size_t iC)`. This is the functions which perform the correct iterations all over the cells, returning the `CellInfo` of index iC. It calls an overloaded function with the same name; by comparing it with respect to the previous one, it takes a further argument, a vector of cells among which it looks for the cell iC. The latter is a recursive function, which for every refined cell calls recursively itself with as argument the vector of the cell's children.

Listing 3.5: Method to iterate over the cells and find the right one.

```
CellInfo* DynamicMesh::find_cell_info(size_t iC) const {
    if (iC >= n_cells())
        ... //error
    if (_n_ref==0) // if the number of refined cells is 0
        return _cells_infos[iC];
    else
        return find_cell_info(iC, _cells_infos);
}
```

```
CellInfo* DynamicMesh::find_cell_info(size_t iC, const std::vector<
    CellInfo*> & c_infos) const {
     size_t count = 0;
     size_t prev_count;
     for (size_t i = 0; i < c_infos.size(); i++){ //for every cell in
         c_infos
          if (count == iC) {
               if (!(c_infos[i]->_has_children)) return c_infos[i];
               else return find_cell_info(iC - count, c_infos[i]->_children
                   );
          } else if (count > iC){
               return find_cell_info(iC - prev_count, c_infos[--i]->
                   _children);
          } else {
               prev_count = count;
               if (!(c_infos[i]->_has_children)) count++;
               else count += c_infos[i]->_has_children;
          }
     }

     if (count > iC) {
          return find_cell_info(iC - prev_count, c_infos[c_infos.size()
              -1]->_children);
     }

     return nullptr;
}
```

3.2.2. The coarsening function

A dynamic mesh, as we explained, should also support the possibility of being coarsened. In this work, as a first step towards this direction, it has been implemented a method to merge children cells coming from the same father, which, therefore, has to be already refined. Therefore, we can go back from the refined to the initial mesh. In upcoming work we will extend further this feature, making the coarsening operation also for the original mesh possible.

Given the already explained structure of the code, and since most of the work is performed by the getter functions - which have to check if each element has children, and properly go through every entity of the mesh - the coarsen method is quite straightforward. It takes as input a `CellInfo` object to be merged with its brother cells, and executes the following steps.

1. It checks if the cell has a father (meaning that it comes from a refined cell).
2. If not, it does nothing.
3. If yes, it deletes all father's children and their children, if they have any.
4. The same procedure is applied to the edges of the father. It checks if each edge needs to be coarsened (i.e. if all its adjacent cells are not refined anymore), and if so, it deletes all its children.

Listing 3.6: Coarsen method

```
void DynamicMesh::coarsen(CellInfo* cell){ // coarsening of 1 level

    if (!cell->_has_father)
        return;

    // - Delete the children cells of the father
    CellInfo* father = cell->_father;
    // for every (father's) child, delete it and all their children,
       recursively
    for (size_t i = 0; i < 3; i++) {
        CellInfo* child = father->_children[i];
        if (!child->_has_children)
            delete child;
        else {
            coarsen(child->_children[0]);
            delete child;
        }
    }
    father->_children.clear();
    father->_has_children = 0;

    // - Decrease _has_children by 3 in all ancestors.
    CellInfo* c = father;
    while (c->_has_father){
        c = c->_father;
        c->_has_children -= 3;
    }

    // - Delete the children edges of the father and fix the indices
    ...
    father->_children_edges.clear();

    // - Coarsen the father's edges, if they do not have anymore refined
```

```
        adjacent cells
    for (auto& e : father->_edges) {
        if (e->_refined_cells == 1)
            edge_coarsen(e);
        else if (e->_refined_cells == 2)
            e->_refined_cells --;
    }
}
```

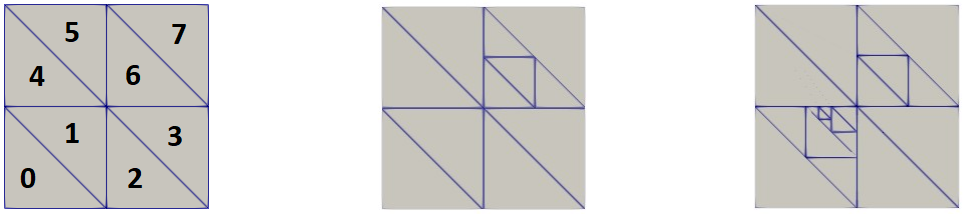
The adaptive algorithm

Finally, the function `DynamicMesh::adapt(vector<size_t> flags)` has been implemented. It takes as input a vector of flags, having the size of the total number of cells in the mesh. The flags denote for every cell if it has to be refined or left the same. In upcoming work, we will include also the coarsen option for each cell.

Listing 3.7: The adapt method

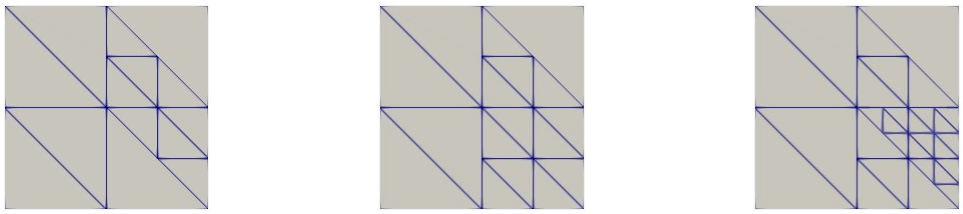
```
void DynamicMesh::adapt(std::vector<size_t> flags){
size_t n_refined = 0; //number of refined cells while executing adapt
   function

    for (size_t i = 0; i < flags.size(); i++){ //iterating over each
       cell
        if (flags[i]==1) {
            refine(find_cell_info(i + n_refined * 3)); // flag's
               elements refer to the cells in order
            n_refined++;
        }
        else if (flags[i]==0) {
            //do nothing
        }
        else{
            std::cout << "ERROR, for now we want flags to be a vector of
                just 0 and 1" << std::endl;
        }
    }
    return;
}
```

Remark. Notice that this function can be improved in upcoming work. Up to now it calls for every flag element (i.e. for every cell) the function `find_cell_info(size_t)`, which goes through all the mesh again. As a consequence this function performs $O(n^2)$



(a) original mesh (b) refined cell 6 (c) refined cells 6, 1, and some children

Figure 3.3: A simple mesh on the left, with visible cells indexing, in the center cell 6 is refined, on the right cell 6, cell 1, one of its children and again one of its further children are refined.



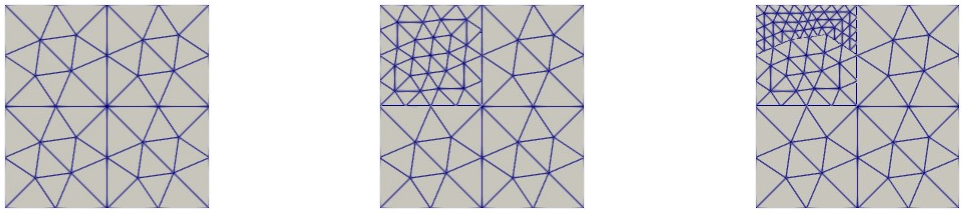
(a) refined cells 6 and 3 (b) refined cells 6, 3 and 2 (c) refined cells 6, 3, 2 and all children of 3

Figure 3.4: Here an example of refined adjacent cells

operations just iterating all over the cells. A much better possibility is to move forward through the flags vector and through all the cells of the mesh at the same time, since they are ordered in the same way.

3.3. Numerical tests

In the last section, our code is finally tested. First of all we show some examples of non-conforming refined meshes, which exploit the methods previously illustrated. In Fig. 3.3 and 3.4 we start with two simple examples of non-conforming refinement, applied to a simple mesh were the indices of the cells are highlighted. First, in In Fig. 3.3 we refine cell 6, then cell 6 plus cell 1 and some of its children and subchildren. As for the second example, in Fig. 3.4, we show the refinement of adjacent cells. Next, in Fig. 3.5 we plot an example with a slightly more sophisticated mesh, taken from HArD::Core library,



(a) mesh1_1.typ2 from HArDCore library

(b) the top left corner is refined

(c) again a top left smaller area is refined

Figure 3.5: mesh1_1.typ2 from HArDCore library is refined

where we refine an area of the mesh, and then again a subarea.

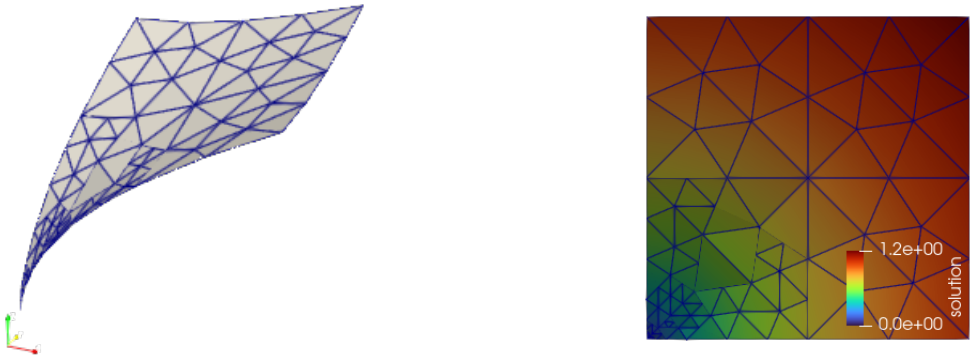
Finally the algorithm 2.1 for the automatic adaptation procedure has been implemented, by exploiting our non-conforming refinement methods. Up to now, it has been used the *true* error in local energy norm as the crucial feature to select which cells of the mesh have to be refined; in the course of our upcoming work, also the version based on a posteriori estimators will be included.

The algorithm, as explained in the previous chapter, takes the 5% of the cells with highest error and refine them, exploiting the function `DynamicMesh::adapt(flags)`, where `flags` marks the selected cells. Differently from the previous chapter, the refinements this time will be non-conforming.

Listing 3.8: Automatic mesh adaptation procedure

```
void automatic_mesh_adaptation(double tol, size_t N_max, DynamicMesh*
    mesh_ptr, ...){

    size_t n = 0;
    double err = tol + 1;

    std::vector<double> l2_errors;
    std::vector<double> en_errors;
    std::vector<size_t> dofs;

    while (err > tol && n < N_max) {

        // 1 - Solve the problem
        ...
        HybridCore hho(mesh_ptr, K + 1, K, use_threads, output);
```

```
HHO_Diffusion model(hho, K, L, kappa, source, BC, exact_solution
    , grad_exact_solution, solver_type, use_threads,output);
...

model.assemble(hho);

UVector u = model.solve(hho);

UVector Uh = hho.interpolate(exact_solution, L, K, 2 * K + 2);

// 2 - Compute the errors and mark the elements

size_t N_top5 = ceil(mesh_ptr->n_cells() * 5.0 / 100.0);
std::vector<double> highest_error(N); // the highest 5% errors
std::vector<size_t> highest_error_indices(N); // the indices of
    their corresponding cell.
...

std::vector<size_t> flags(mesh_ptr->n_cells());

for (size_t iT = 0; iT < flags.size(); iT++){ //for every cell I
     compute the error
    double local_error = model.EnergyNorm_cell(hho, u - Uh, iT);
    ...
    double min = minimum(highest_error, index); //index is a
        reference where you get the index of the minimum within
        the vector highest_error
    if (local_error > min){
        highest_error[i] = local_error;
        highest_error_indices[index] = iT;
    }
}

for (size_t k : highest_error_indices)
    flags[k] = 1;

// store errors and dofs at each iteration
double L2error = hho.L2norm(u - Uh) / hho.L2norm(Uh);
double EnergyError = model.EnergyNorm(hho, u - Uh) / model.
    EnergyNorm(hho, Uh);
size_t nbedgedofs = model.get_ntotal_edge_dofs();
l2_errors.push_back(L2error);
en_errors.push_back(EnergyError);
dofs.push_back(nbedgedofs);
```



(a) (b)

Figure 3.6: Exact solution for Fichera corner problem (shown on an adaptively refined grid). We can observe a singularity in the origin.

```
        // 3 - refine the marked elements

        mesh_ptr->adapt(flags);

        n++;
        err = L2error;

    }
}
```

Finally we are ready to perform some convergence tests, by testing our methods on a diffusion problem with irregular solution, namely the Fichera corner problem already mentioned in Chapter 2, whose solution is:

$$u(x_1, x_2) = \sqrt[4]{x_1^2 + x_2^2},$$

which is singular in the origin (Fig. 3.6).

The convergence estimates are, therefore, not proven to be true, and indeed we showed in Section 2.3 that they do not hold for a uniformly refined mesh sequence. Here, we show that, by exploiting the adaptation algorithm with our implemented methods for non-conforming refinement of the mesh, optimal orders of convergence are recovered. Let us recall the optimal orders of convergence: $N_{\text{dof}}^{\frac{(k+2)}{d}}$ for energy-norm and $N_{\text{dof}}^{\frac{(k+1)}{d}}$ for L^2-norm. The convergence results are displayed in figures 3.7, 3.8, 3.9, 3.10, with different

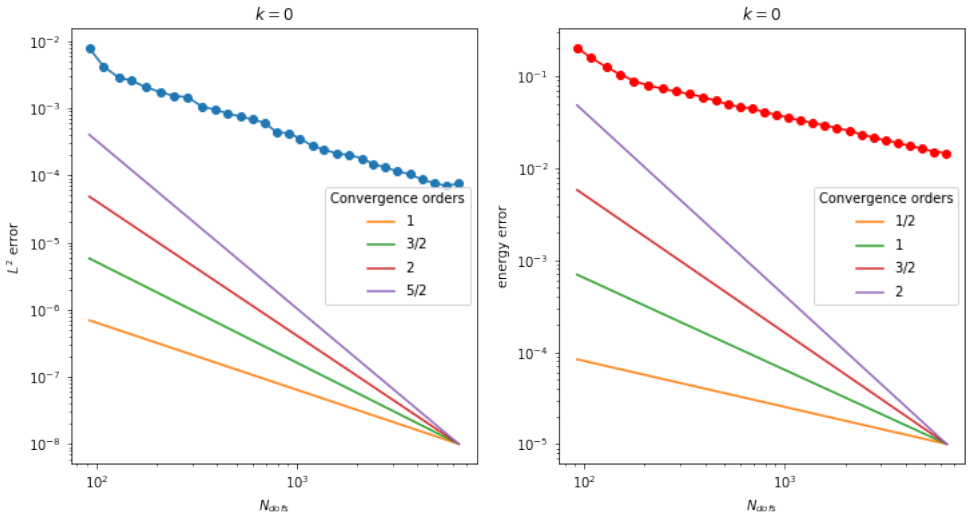


Figure 3.7: Error vs. N_{dof} for the test case of Fichera problem with polynomial degree 0. The expected convergence rate is N_{dof}^{-1} for the error in L^2 norm, and $N_{\text{dof}}^{-1/2}$ for the error in energy norm.

polynomial degrees.

Let us conclude by showing the refined mesh (mesh1_1.typ2 displayed in figure 3.5) at a different number of iterations of the adaptation algorithm 2.1, still applied to the Fichera problem. See Fig. 3.11 for further details.

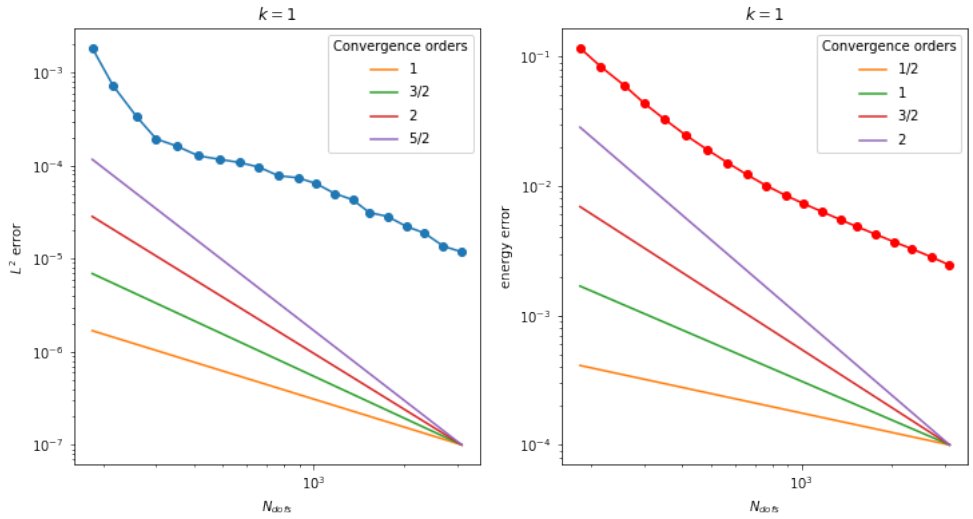


Figure 3.8: Error vs. N_{dof} for the Fichera solution with polynomial degree 1. The expected convergence rate is $N_{\text{dof}}^{-3/2}$ for the error in L^2 norm, and N_{dof}^{-1} for the error in energy norm.

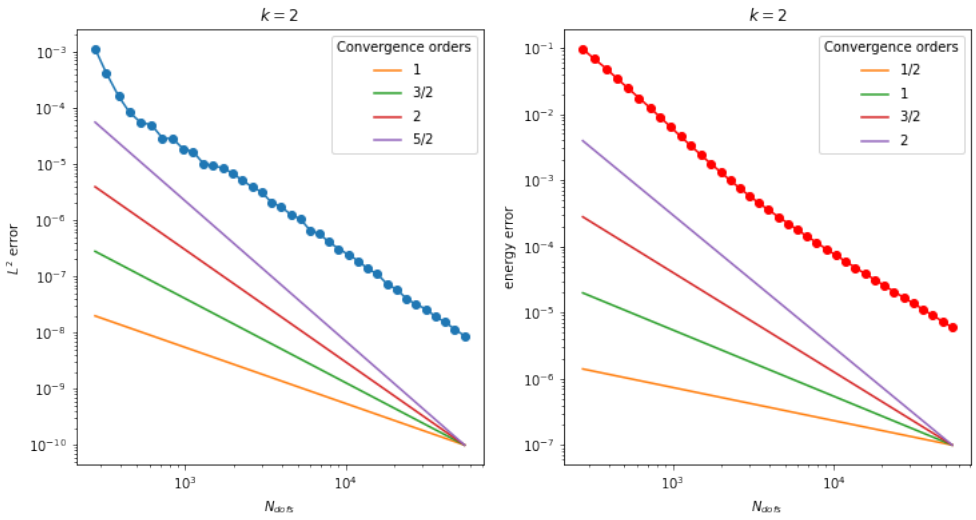


Figure 3.9: Error vs. N_{dof} for the Fichera solution with polynomial degree 2. The expected convergence rate is N_{dof}^{-2} for the error in L^2 norm, and $N_{\text{dof}}^{-3/2}$ for the error in energy norm.

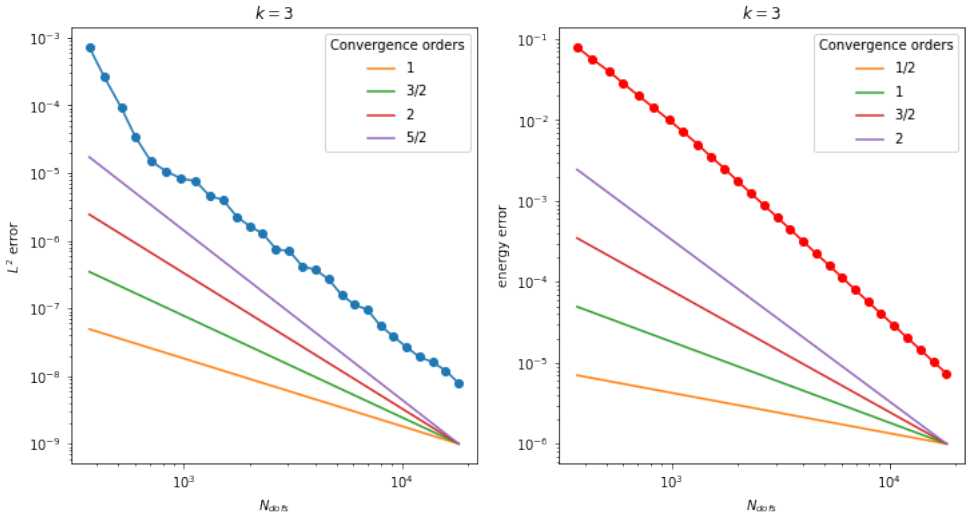
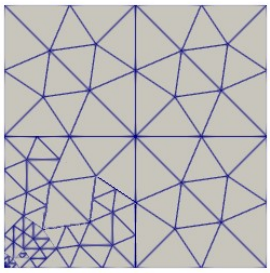
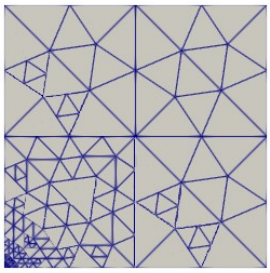


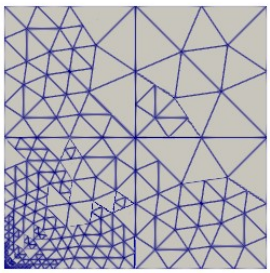
Figure 3.10: Error vs. N_{dof} for the Fichera solution with polynomial degree 3. Expected convergence rate: $N_{\text{dof}}^{-5/2}$ for the L^2 error, N_{dof}^{-2} energy error.



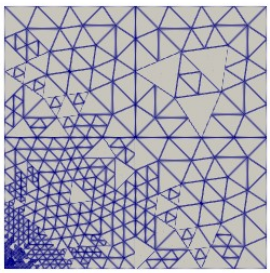
(a) 5 iterations



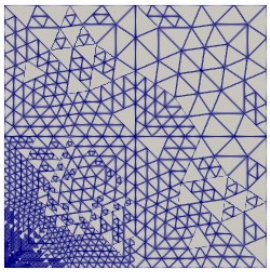
(b) 10 iterations



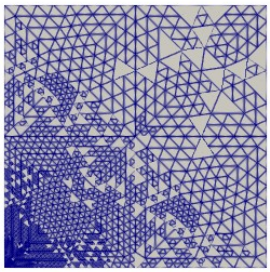
(c) 15 iterations



(d) 20 iterations



(e) 25 iterations



(f) 30 iterations

Figure 3.11: Different number of iterations for the adaptive refinement algorithm with mesh mesh1_1.typ2

4 | Conclusions and future developments

As we showed in the last section, a non-conforming adaptivity of the mesh is able to recover the optimal convergence rates also for poorly regular solutions. The novelty introduced in our work, namely the non conformity of the refinement, can potentially significantly reduce the computational cost required by an adaptation procedure, since it needs to re-build the mesh just locally at every refinement operation. It suits perfectly problems where physical phenomena occur in small areas of the domain, or problems which result in poorly regular solutions, like interface problems.

Future research can enhance our work on several grounds.
First of all, the next goal could be to go further with the adaptation procedure, focusing on incorporating also the coarsening feature, in such a way that, where the numerical scheme turns out to have a low error, the mesh can be coarsened in order to save computational effort. Another interesting possibility is to make the adaptation of the mesh driven by a posteriori-estimators, instead of the true errors, so that it can be used also in more complex problems, whose solution is unknown. Moreover, the computational effort of our method could be explored more deeply, looking also for smart ways to avoid jumps in the memory space used by our algorithm.

Bibliography

[1] C. Carstensen, M. Feischl, M. Page, and D. Praetorius. Axioms of adaptivity. *Computers & Mathematics with Applications*, 67(6):1195–1253, 2014.

[2] D. A. Di Pietro and J. Droniou. The hybrid high-order method for polytopal meshes. *Design, analysis, and applications*, 19, 2019.

[3] D. A. Di Pietro and A. Ern. A hybrid high-order locking-free method for linear elasticity on general meshes. *Computer Methods in Applied Mechanics and Engineering*, 283:1–21, 2015.

[4] D. A. Di Pietro and R. Specogna. An a posteriori-driven adaptive mixed high-order method with application to electrostatics. *Journal of Computational Physics*, 326: 35–55, 2016.

[5] D. A. Di Pietro and R. Tittarelli. An introduction to hybrid high-order methods. In *Numerical methods for PDEs*, pages 75–128. Springer, 2018.

[6] D. A. Di Pietro, A. Ern, and S. Lemaire. An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators. *Computational Methods in Applied Mathematics*, 14(4):461–472, 2014.