Algorithms for the implementation of adaptive isogeometric methods using hierarchical splines

Eduardo M. Garau[a,*], Rafael Vázquez[b,c]

[a]*Instituto de Matemática Aplicada del Litoral, UNL, CONICET, FIQ, Santa Fe, Argentina*
[b]*Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes' (CNR), Pavia, Italy*
[c]*Institute of Mathematics, École Polytechnique Fédérale de Lausanne, Station 8, 1015 Lausanne, Switzerland*

Abstract

In this article we introduce all the ingredients to develop adaptive isogeometric methods based on hierarchical splines. In particular, we give precise definitions of local refinement and coarsening that, unlike previously existing methods, can be understood as the inverse of each other. We also define simple and intuitive data structures for the implementation of hierarchical splines, and algorithms for refinement and coarsening that take advantage of local information. We complete the paper with some simple numerical tests to show the performance of the data structures and algorithms, that have been implemented in the open-source Octave/Matlab code GeoPDEs.

Keywords: Isogeometric analysis, adaptive methods, hierarchical splines, local refinement, coarsening

1. Introduction

The use of high order spline spaces for the numerical discretization of partial differential equations has been increased and spread due to the emergence of the isogeometric analysis (IGA) techniques [1, 2]. Introduced originally to enhance the interoperability between computer aided design (CAD) and finite element softwares, the main idea behind IGA is to use splines or rational splines (NURBS) functions, which are the standard in CAD, both for geometry representation and for the computation of the discrete solution to the equation. A state-of-the-art review on the existing mathematical results can be found in [3].

One of the most active research topics on IGA is the development of adaptive methods for local refinement and coarsening. These methods require the use of spline spaces that go beyond the tensor product structure of B-splines [4, 5], and several alternatives have been already proposed and tested in the IGA community, such as hierarchical splines, T-splines, LR-splines or PHT-splines. Among them, hierarchical splines are probably the easiest to define and to implement for their use in IGA [6], thanks to their multilevel structure: first one introduces a sequence of B-spline spaces of different levels and a sequence of subdomains, which determine the hierarchical mesh, then the set of active functions in each level is decided by a simple check on their support compared to the subdomains. This simplicity, together with the flexibility in the choice of the refined subdomains, has favored their application by many different research groups, see for example [7, 8, 9, 10].

However, when facing the implementation of adaptive IGA methods with hierarchical splines, most of the works we are aware of [11, 12, 13, 14] fail to reflect this simplicity and/or flexibility in their algortihms and data structures. Entering into the details, the data structures in [11] constrain to refine on disjoint (closed) hyperrectangular regions, which is not flexible enough for adaptive refinement. The data structures in [12] are probably the clearest ones, but their algorithms neglect local information for the update of active functions and elements, in practice requiring a global recomputation of the data structures even when local

*Corresponding author
Email addresses: egarau@santafe-conicet.gov.ar (Eduardo M. Garau), vazquez@imati.cnr.it (Rafael Vázquez)

refinement is performed. Moreover, they only detail the implementation in the case of a uniform knot vector in each level, while the same structures can be used for the non-uniform case, and even for degree elevation. Instead, the update of active functions in [13] is done based on local information, but their data structures are unnecessarily complicated, as they make use both of active vertices and elements, while for the definition only elements are needed. A completely different data structure to represent the subdomains, based on a binary subdivision tree, is given in [14] as a generalization of the two-dimensional implementation in [15]. Although the use of a binary tree can be computationally efficient, the relation between the tree and the hierarchical mesh is hardly intuitive, and in fact the same mesh can be represented with many different binary trees. Moreover, the way to perform matrix assembly in [14] is far from being optimal, as we will see in §7.

Our first goal in this work is to introduce a new set of data structures and algorithms for a clear and intuitive implementation of adaptive IGA methods with hierarchical splines, and to demonstrate how they work with the help of the Octave/Matlab package GeoPDEs [16, 17]. The implementation is based on two data structures: one for the hierarchical mesh, with the list of active elements on each level, and one for the hierarchical basis, with the list of active functions. These two simple structures provide all the necessary information required by the adaptive IGA methods, and they are valid for standard hierarchical splines [6], for the simplified hierarchical basis introduced in [18], and also for truncated hierarchical splines [19, 20].

Our second goal is to introduce a rigorous definition of coarsening, and the corresponding algorithms. In fact, although all the mentioned works basically agree on the way to perform refinement, either directly refining marked elements or refining elements within the support of marked functions, there is no agreement on the way to perform coarsening. In particular, the coarsening strategies in [12, 13] are very agressive, in the sense that they allow to reactivate coarse elements that have been refined several times. Similar to [21] and in part inspired by their work, our definition of coarsening can be proved to be the inverse of refinement, but with the advantage that our coarsening algorithms are less restrictive than the ones in [21], and they also work for high order splines.

The paper is organized as follows. In §2 we introduce some basic definitions and properties about hierarchical splines, and in §3 we recall how they are used in IGA, with particular attention to the algorithm of matrix assembly. In §4 we introduce rigorous definitions of the refinement and coarsening procedures. The data structures to implement adaptive isogeometric methods based on hierarchical splines are detailed in §5, and are applied in §6 to explain the refinement and coarsening algorithms. The implementation of these algorithms in the Octave/Matlab package GeoPDEs is briefly explained in §7, along with several numerical tests to show its performance.

For clarity and simplicity, throughout the paper we restrict ourselves to the case of hierarchical B-splines. However, all the algorithms in this paper work analogously for any kind of hierarchical space that satisfies the conditions in [20], provided that the structures and functions described in §5 can be defined.

2. The basics about hierarchical splines

We give in this section some basic definitions about hierarchical splines, defined as in [6, 22]. In the following we will assume that the spaces are defined on the closed parametric domain $\widehat{\Omega} = [0,1]^d \subset \mathbb{R}^d$.

2.1. Underlying sequence of tensor product spline spaces

We consider a given sequence $\{\mathcal{S}_\ell\}_{\ell\in\mathbb{N}_0}$ of tensor product d-variate spline spaces such that

$$\mathcal{S}_0 \subset \mathcal{S}_1 \subset \mathcal{S}_2 \subset \mathcal{S}_3 \subset \ldots, \tag{1}$$

which are determined by their degree and their knot vectors. For $\ell \in \mathbb{N}_0$, the B-spline basis corresponding to $\mathcal{S}_\ell$ is denoted by $\mathcal{B}_\ell := \{\beta_{i,\ell} \mid i = 1,\ldots,N_\ell\}$, where N_ℓ is the dimension of the space $\mathcal{S}_\ell$. Furthermore, we denote by $\mathcal{Q}_\ell$ the Cartesian mesh associated to $\mathcal{B}_\ell$, and we say that $Q \in \mathcal{Q}_\ell$ is a *cell of level ℓ*. We note that, as in [23], we assume that the cells are closed sets.

B-splines possess several important properties, such as non-negativity, partition of unity, local linear independence and local support, that make them suitable for design and analysis, see [2, 4, 5] for details.

In this work we will make extensive use of the *two-scale relation with non-negative coefficients*, which says that B-splines of level ℓ can be written as linear combinations of B-splines of level $\ell+1$ with non-negative coefficients, that is:

$$\beta_{i,\ell} = \sum_{k=1}^{N_{\ell+1}} c_{k,\ell+1}(\beta_{i,\ell})\, \beta_{k,\ell+1}, \qquad \forall\, \beta_{i,\ell} \in \mathcal{B}_\ell, \tag{2}$$

with $c_{k,\ell+1}(\beta_{i,\ell}) \geq 0$. We notice that, due to the local linear independence of B-splines, only a limited number of the coefficients $c_{k,\ell+1}(\beta_{i,\ell})$ are different from zero. We will say that $\beta_{k,\ell+1}$ is a *child* of $\beta_{i,\ell}$ if $c_{k,\ell+1}(\beta_{i,\ell}) \neq 0$, and denote by $\mathcal{C}(\beta_{i,\ell}) \subset \mathcal{B}_{\ell+1}$ the set of children of $\beta_{i,\ell}$. Reciprocally, we define the parents of a basis function $\beta_{k,\ell+1} \in \mathcal{B}_{\ell+1}$ as $\mathcal{P}(\beta_{k,\ell+1}) := \{\beta_{i,\ell} \in \mathcal{B}_\ell \mid \beta_{k,\ell+1} \in \mathcal{C}(\beta_{i,\ell})\}$. In a similar way, for a cell $Q \in \mathcal{Q}_\ell$ we say that $Q' \in \mathcal{Q}_{\ell+1}$ is a *child* of Q, or equivalently that Q is a *parent* of Q', if $Q' \subset Q$.

Due to the nestedness of the tensor product spline spaces (1), one can consider the inclusion map between two consecutive levels $I_\ell^{\ell+1} : \mathcal{S}_\ell \longrightarrow \mathcal{S}_{\ell+1}$. Considering the particular choice of the bases $\mathcal{B}_\ell$ and $\mathcal{B}_{\ell+1}$, this inclusion can be expressed in the form of a matrix, that we denote by $C_\ell^{\ell+1}$, whose entries are given by the coefficients of the two-scale relation, that is

$$(C_\ell^{\ell+1})_{ki} = c_{k,\ell+1}(\beta_{i,\ell}), \quad \text{ for } i = 1,\ldots,N_\ell, \quad k = 1,\ldots,N_{\ell+1}.$$

For tensor product B-splines, the matrix $C_\ell^{\ell+1}$ is computed as the Kronecker tensor product of the analogous matrices corresponding to the univariate case.

Successively applying the two-scale relation (2), for $m \in \mathbb{N}$ we obtain the general version

$$\beta_{i,\ell} = \sum_{k=1}^{N_{\ell+m}} c_{k,\ell+m}(\beta_{i,\ell}) \beta_{k,\ell+m}, \qquad \forall\, \beta_{i,\ell} \in \mathcal{B}_\ell, \tag{3}$$

and the corresponding matrix operator $C_\ell^{\ell+m} = C_{\ell+m-1}^{\ell+m} \cdots C_{\ell+1}^{\ell+2} C_\ell^{\ell+1}$, which relates functions of non-consecutive levels.

2.2. Hierarchical B-splines

For the definition of hierarchical B-splines we follow [6, 22].

Definition 1. *[Hierarchy of subdomains] For $n \in \mathbb{N}$, we say that the set $\mathbf{\Omega}_n := \{\Omega_0, \Omega_1, \ldots, \Omega_n\}$ is a hierarchy of subdomains of depth n if*

$$\widehat{\Omega} = \Omega_0 \supset \Omega_1 \supset \cdots \supset \Omega_{n-1} \supset \Omega_n = \emptyset, \tag{4}$$

and each subdomain Ω_ℓ is the union of cells of level $\ell - 1$.

Definition 2. *[Standard hierarchical basis] Let $\{\mathcal{S}_\ell\}_{\ell \in \mathbb{N}_0}$ be a sequence of spaces like* (1) *with the corresponding B-spline bases $\{\mathcal{B}_\ell\}_{\ell \in \mathbb{N}_0}$, and $\mathbf{\Omega}_n := \{\Omega_0, \Omega_1, \ldots, \Omega_n\}$ a hierarchy of subdomains of depth n. We define the hierarchical B-spline basis $\mathcal{H} \equiv \mathcal{H}(\mathbf{\Omega}_n)$ by taking $\mathcal{H} := \mathcal{H}_{n-1}$ in the following recursive algorithm:*

$$\begin{cases} \mathcal{H}_0 & := & \mathcal{B}_0, \\ \mathcal{H}_{\ell+1} & := & \{\beta \in \mathcal{H}_\ell \mid \operatorname{supp}\beta \not\subset \Omega_{\ell+1}\} \cup \\ & & \{\beta \in \mathcal{B}_{\ell+1} \mid \operatorname{supp}\beta \subset \Omega_{\ell+1}\}, \quad \ell = 0,\ldots,n-2. \end{cases}$$

The hierarchical spline basis is associated to an underlying *hierarchical mesh* $\mathcal{Q} \equiv \mathcal{Q}(\mathbf{\Omega}_n)$, given by

$$\mathcal{Q} := \bigcup_{\ell=0}^{n-1} \{Q \in \mathcal{Q}_\ell \mid Q \subset \Omega_\ell \wedge Q \not\subset \Omega_{\ell+1}\}.$$

In the following we say that Q is an *active cell* (or *active element*) if $Q \in \mathcal{Q}$, and it is an *active cell of level* ℓ if $Q \in \mathcal{Q}_\ell \cap \mathcal{Q}$. We will also say that Q is a *deactivated cell of level* ℓ if $Q \in \mathcal{Q}_\ell$ and $Q \subset \Omega_{\ell+1}$.

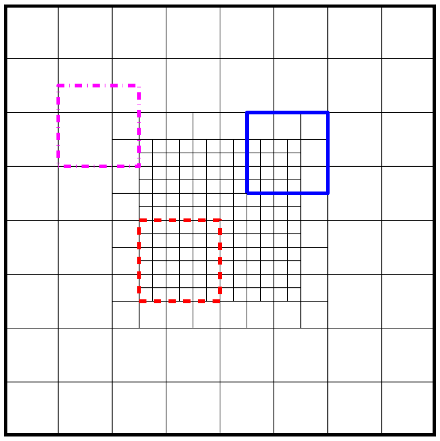


Figure 1: Example of biquadratic B-splines of maximum smoothness over a three-level mesh. Three functions of level 1 are shown by highlighting their supports. One function is active (solid blue line), one is deactivated (red dashed line), and one is passive (purple dashed-dotted line).

Analogously, we say that β is an *active (basis) function* if $\beta \in \mathcal{H}$, it is an *active function of level* ℓ if $\beta \in \mathcal{F}_\ell^A := \mathcal{H} \cap \mathcal{B}_\ell$, and it is a *deactivated function of level* ℓ if $\beta \in \mathcal{F}_\ell^D := \mathcal{H}_\ell \setminus \mathcal{H}_{\ell+1}$, i.e., $\operatorname{supp} \beta \subset \Omega_{\ell+1}$. Moreover, $\mathcal{H}_\ell \cap \mathcal{B}_\ell$ is the union of active and deactivated functions of level ℓ. Note that a function of level ℓ is active if all the active cells within its support are of level ℓ or higher, and at least one of such cells is actually of level ℓ. A function is deactivated when all the cells of its level within the support are deactivated. Basis functions of level ℓ whose support is not contained in Ω_ℓ are neither active nor deactivated, and will be referred to as *passive functions*. In Figure 1 we show, in an example of biquadratic hierarchical splines, a basis function of each type.

It will be helpful, specially to write some equations in matrix form, to give a notation for the number of these functions. We will denote by $N := \#\mathcal{H}$ the dimension of the hierarchical space, and by $N_\ell^A := \#(\mathcal{H} \cap \mathcal{B}_\ell)$ and $N_\ell^D := \#(\mathcal{H}_\ell \setminus \mathcal{H}_{\ell+1})$ the number of active and deactivated functions of level ℓ, respectively. Sometimes we will also make use of $N_{0:\ell}^A = \sum_{k=0}^{\ell} N_k^A$, the number of active functions up to level ℓ.

Finally, we remark that, unlike in the tensor product case, the basis functions of the hierarchical space do not form a partition of unity, although the unity belongs to the space. That is, there exist coefficients a_β such that

$$\sum_{\beta \in \mathcal{H}} a_\beta \beta(\widehat{\mathbf{x}}) = 1, \qquad \text{for } \widehat{\mathbf{x}} \in \widehat{\Omega}, \tag{5}$$

and it can be proved that $a_\beta \geq 0$.

2.3. A simplified hierarchical B-spline space

The hierarchical B-splines of the previous subsection are the ones traditionally used in isogeometric analysis [6, 12], and basis functions are activated according to their supports. In [18] the authors introduced a different hierarchical space, with the same approximation properties as the one in the previous section but with lower dimension. In this space, the decision to activate basis functions is based not only on the support, but also on the relation between functions of different levels.

Definition 3. *[Simplified hierarchical basis] Let $\{\mathcal{S}_\ell\}_{\ell \in \mathbb{N}_0}$ be a sequence of spaces like* (1) *with the corresponding B-spline bases $\{\mathcal{B}_\ell\}_{\ell \in \mathbb{N}_0}$, and $\boldsymbol{\Omega}_n := \{\Omega_0, \Omega_1, \ldots, \Omega_n\}$ a hierarchy of subdomains of depth n. We define the simplified hierarchical basis $\widetilde{\mathcal{H}} := \widetilde{\mathcal{H}}_{n-1}$ computed with the following recursive algorithm:*

$$\begin{cases} \widetilde{\mathcal{H}}_0 := \mathcal{B}_0, \\ \widetilde{\mathcal{H}}_{\ell+1} := \{\beta \in \widetilde{\mathcal{H}}_\ell \mid \operatorname{supp} \beta \not\subset \Omega_{\ell+1}\} \cup \bigcup\limits_{\substack{\beta \in \widetilde{\mathcal{H}}_\ell \\ \operatorname{supp} \beta \subset \Omega_{\ell+1}}} \mathcal{C}(\beta), \quad \ell = 0, \ldots, n-2. \end{cases}$$

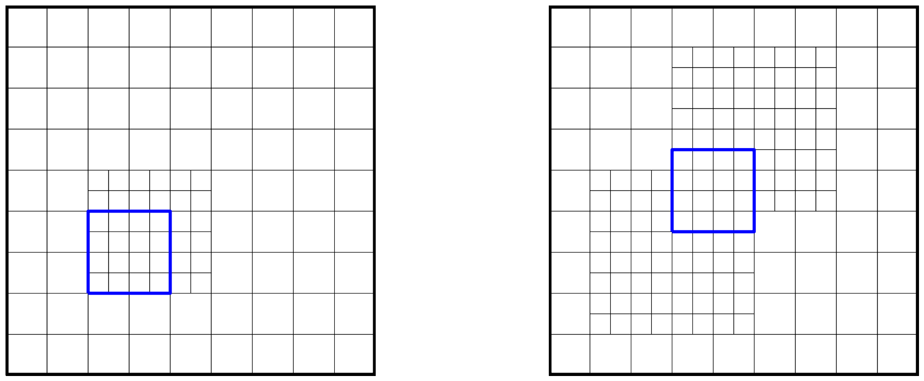


Figure 2: Some examples of two-level meshes for bicubic splines of maximum smoothness. In both cases, the highlighted B-splines of level 1 have support included in Ω_1, but they are not children of any deactivated B-spline of level 0, and thus, they belong to the hierarchical basis $\mathcal{H}$ but not to the simplified basis $\widetilde{\mathcal{H}}$.

Unlike in the algorithm of Definition 2, where a basis function of level $\ell+1$ is added if its support is contained in $\Omega_{\ell+1}$, in this simplified space, basis functions of level $\ell+1$ are added only if they are children of a deactivated function of level ℓ; see the examples in Figure 2. This may lead to simpler refinement schemes, specially for a posteriori estimators which are based on the basis functions and not on the elements [24].

An interesting property of this basis is that now the coefficients for writing the unity are strictly positive. That is, we have

$$\sum_{\beta \in \widetilde{\mathcal{H}}} a_\beta \beta(\widehat{\mathbf{x}}) = 1, \qquad \text{for } \widehat{\mathbf{x}} \in \widehat{\Omega},$$

with $a_\beta > 0$ (see [18, Theorem 5.2]).

Notice that, given the hierarchy of subdomains $\mathbf{\Omega}_n$, the underlying hierarchical mesh is the same for both bases $\mathcal{H}$ and $\widetilde{\mathcal{H}}$. Moreover, the evaluation of the basis functions will be done exactly in the same way, and from the point of view of the implementation the only difference is the computation of the set of active functions during refinement and coarsening. For this reason, in the following we will refer to both bases with the generic notation $\mathcal{H}$, and only use the notation $\widetilde{\mathcal{H}}$ when it is necessary to remark the difference between the two bases.

Remark 1. *In many cases the simplified hierarchical basis coincides with the one obtained by "quasi-hierarchical" refinement in [21] (see also [25]), where refinement is applied deactivating marked functions and activating their children. However, the space in [21] is not uniquely determined by the sequence of spaces* (1) *and the hierarchy of subdomains* (4)*, like in the two examples of Figure 3: both sets of active functions can be obtained with their refinement strategy, but the corresponding mesh only has active elements of level* 1*, which for the simplified hierarchical basis implies that only B-splines of level* 1 *are active. Moreover, for high order splines the refinement in [21] may lead to linear dependencies, like in the example of Figure 3 (right), where all the children of the active function of level* 0 *are also active.*

2.4. Truncated hierarchical B-splines

Truncated hierarchical B-splines (or THB-splines) were introduced and analysed in [19, 20]. THB-splines represent an alternative basis for the space of hierarchical splines, that recovers the partition of unity [19, Theorem 10] and reduces the support of the basis functions, therefore reducing the interaction between them. In particular, this leads to sparser matrices in isogeometric analysis. Moreover, the better properties of THB-splines with respect to the standard basis make them more appropriate to develop the mathematical theory of approximation [23] and adaptivity [26].

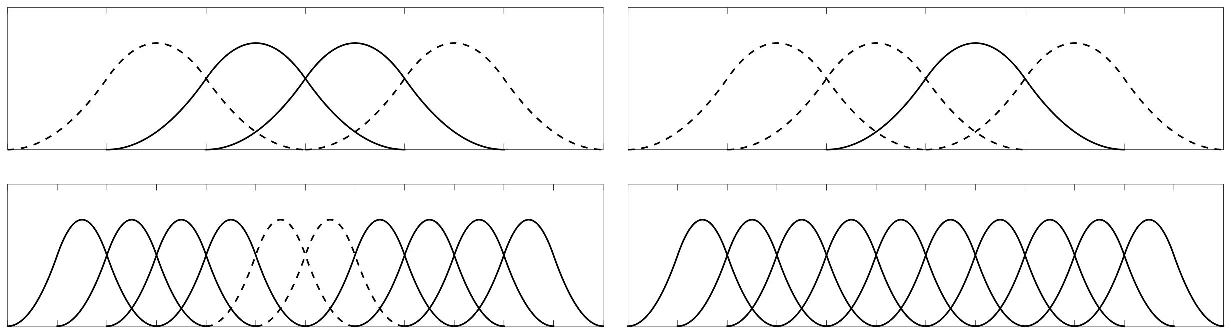


Figure 3: Two examples for quadratic splines like in [21] that are not allowed with the simplified hierarchical space. In the example on the left two functions of level 0 are deactivated (dashed lines), and their children are active, which gives the same mesh as for the B-spline space of level 1. Deactivating another function of level 0, as in the example on the right, changes the set of active functions without changing the mesh, and it also adds a linear dependency.

The definition of THB-splines requires first the definition of the truncation operator $\mathrm{trunc}_{\ell+1}$. This consists in removing in the two-scale relation (2) the contribution of active and deactivated functions of level $\ell+1$, that we recall to be the functions in $\mathcal{H}_{\ell+1} \cap \mathcal{B}_{\ell+1}$. Since $\mathrm{trunc}_{\ell+1}$ is a linear operator in $\mathcal{S}_\ell$, it is sufficient to define it for the basis functions:

$$\mathrm{trunc}_{\ell+1}(\beta_{i,\ell}) := \sum_{k=1}^{N_{\ell+1}} c^{\tau}_{k,\ell+1}(\beta_{i,\ell})\beta_{k,\ell+1}, \qquad \text{for } \beta_{i,\ell} \in \mathcal{B}_\ell,$$

where the coefficients are

$$c^{\tau}_{k,\ell+1}(\beta_{i,\ell}) = \begin{cases} 0 & \text{if } \beta_{k,\ell+1} \in \mathcal{H}_{\ell+1} \cap \mathcal{B}_{\ell+1}, \\ c_{k,\ell+1}(\beta_{i,\ell}) & \text{otherwise.} \end{cases}$$

Now, following [19] we define the truncated hierarchical basis $\mathcal{T} := \mathcal{T}_{n-1}$ computed by the following recursive algorithm:

$$\begin{cases} \mathcal{T}_0 & := & \mathcal{B}_0, \\ \mathcal{T}_{\ell+1} & := & \{\mathrm{trunc}_{\ell+1}(\beta) \mid \beta \in \mathcal{T}_\ell \wedge \operatorname{supp}\beta \not\subset \Omega_{\ell+1}\} \cup \\ & & \{\beta \in \mathcal{B}_{\ell+1} \mid \operatorname{supp}\beta \subset \Omega_{\ell+1}\}, \qquad \ell = 0, \ldots, n-2. \end{cases}$$

It is easy to check that $\operatorname{span}\mathcal{H} = \operatorname{span}\mathcal{T}$, see [19].

Notice that, after truncation, a basis function of level ℓ is not a function of $\mathcal{B}_\ell$ anymore, and it has to be expressed in terms of the basis $\mathcal{B}_{\ell'}$, where $\ell' > \ell$ is the level of the finest function that truncates it. Therefore, to evaluate truncated functions we introduce, analogously to the matrices $C_\ell^{\ell+1}$, the new matrices $C_\ell^{\ell+1,\tau}$ that collect the coefficients $c^{\tau}_{k,\ell+1}(\beta_{i,\ell})$, which is equivalent to set to zero in $C_\ell^{\ell+1}$ the rows corresponding to active and deactivated functions of level $\ell+1$. Moreover, analogously to the matrix $C_\ell^{\ell+m}$ in §2.1, we define

$$C_\ell^{\ell+m,\tau} = C_{\ell+m-1}^{\ell+m,\tau} \ldots C_{\ell+1}^{\ell+2,\tau} C_\ell^{\ell+1,\tau}.$$

Notice also that the restriction of a truncated basis function of level ℓ to an active element of the same level remains unaffected. Similarly, the restriction to an active element of level $k \geq \ell$ is not affected when truncating with functions of level greater than k, and the truncated function restricted to such element can be computed using the matrix $C_\ell^{k,\tau}$, even if it is truncated by finer functions.

Finally, we remark that truncation can be also applied to the hierarchical basis $\widetilde{\mathcal{H}}$, setting to zero some rows of the matrix for the two-scale relation in the same way.

3. Isogeometric analysis with hierarchical splines

As already mentioned in the introduction, isogeometric methods with hierarchical splines have been introduced in [6] and investigated in several works. In this section we address, without entering into details, some issues to be considered for the implementation.

3.1. The general setting

Recalling that $\widehat{\Omega} = [0,1]^d$, we assume that the domain is given by a certain parametrization $\mathbf{F} : \widehat{\Omega} \longrightarrow \Omega \subset \mathbb{R}^r$, with $r \geq d$. Then, we consider the discrete space

$$W = \operatorname{span}\{\beta \circ \mathbf{F}^{-1} \mid \beta \in \mathcal{H}\} = \operatorname{span}\{\beta \circ \mathbf{F}^{-1} \mid \beta \in \mathcal{T}\}.$$

In general, we want to solve a discrete variational problem: find $u \in V$ such that

$$a(u,v) = \langle f, v\rangle, \quad \forall v \in V,$$

where $a(\cdot,\cdot)$ and f are a given bilinear form and a linear operator, respectively, and the choice of the discrete space $V \subset W$ will usually depend on the boundary conditions of the problem.

As it is standard in isogeometric analysis, one can make use of the isogeometric paradigm, and consider that the parametrization $\mathbf{F}$ is given in terms of hierarchical splines in $\mathcal{H}$, associating a control point $\mathbf{C}_\beta \in \mathbb{R}^r$ to each basis function:

$$\mathbf{F}(\widehat{\mathbf{x}}) = \sum_{\beta\in\mathcal{H}} \mathbf{C}_\beta \beta(\widehat{\mathbf{x}}), \qquad \forall \widehat{\mathbf{x}} \in \widehat{\Omega}.$$

Actually, most of the times the parametrization $\mathbf{F}$ will be expressed in terms of the tensor product splines of the coarsest level, $\mathcal{B}_0$.

3.2. Matrix assembly

One of the main issues when implementing isogeometric methods with hierarchical splines is the assembly of the matrix, since the integrals involve active functions of different levels, and possibly acting on active elements from a third different level. For the assembly we follow the method proposed by [12] and [9], also used by [7] for THB-splines, which is based on the two-scale relation (3). We explain the method for the mass matrix, but the idea applies to any other relevant matrix, and also to the vector of the right-hand side. For simplicity we write the computations in the parametric domain $\widehat{\Omega}$, but since the two-scale relation is not affected by the composition with $\mathbf{F}^{-1}$, the method works exactly the same in the physical domain $\Omega = \mathbf{F}(\widehat{\Omega})$.

Let us first denote the hierarchical basis functions in $\mathcal{H}$ by

$$\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_N\}.$$

Thus, for each m, $1 \leq m \leq N$, there exist unique integers k_m and ℓ_m such that $\eta_m = \beta_{k_m,\ell_m}$. We will assume that the functions in $\mathcal{H}$ are ordered by levels, and inside each level ℓ they follow the same ordering as in $\mathcal{B}_\ell$. That is, if $m < m'$ then $\ell_m \leq \ell_{m'}$, and if the two levels are equal then $k_m < k_{m'}$.

To compute one entry of the mass matrix $M = (M_{ij})_{i,j=1,\ldots,N}$, one must compute the following integral, which involves only active elements where the functions do not vanish:

$$M_{ij} = \int_{\widehat{\Omega}} \eta_j\, \eta_i = \sum_{Q\in\mathcal{Q}} \int_Q \eta_j\, \eta_i = \sum_{\ell=\max\{\ell_i,\ell_j\}}^{n-1} \sum_{Q_\ell\in\mathcal{Q}_\ell\cap\mathcal{Q}} \int_{Q_\ell} \eta_j\, \eta_i.$$

Recalling that we write $\eta_m = \beta_{k_m,\ell_m}$, and denoting by $c^\ell_{km} = c_{k,\ell}(\beta_{k_m,\ell_m})$ the coefficients in (3) to express η_m in terms of basis functions of level ℓ, we can replace in the integrals each basis function with such expression. Taking the constants out of the integral and reordering the sums, we obtain

$$M_{ij} = \sum_{\ell=\max\{\ell_i,\ell_j\}}^{n-1} \sum_{k=1}^{N_\ell} \sum_{k'=1}^{N_\ell} c^\ell_{ki} \left(\sum_{Q_\ell\in\mathcal{Q}_\ell\cap\mathcal{Q}} \int_{Q_\ell} \beta_{k,\ell}\beta_{k',\ell} \right) c^\ell_{k'j}. \tag{6}$$

After the arrangements, we see that it is only needed to compute, in the active elements of level ℓ, the integrals involving the tensor product functions of that level, independently on whether they are active, deactivated or passive. The computation of these integrals should be easily available in any IGA software.

The only missing part is the computation of the coefficients in an efficient way, that can be done rewriting (6) in matrix form. Let us denote by M_ℓ the matrix obtained when computing the integrals for functions of level ℓ, that is,

$$(M_\ell)_{kk'} := \sum_{Q_\ell \in \mathcal{Q}_\ell \cap \mathcal{Q}} \int_{Q_\ell} \beta_{k,\ell} \beta_{k',\ell},$$

and let us denote by $\mathbf{c}_m^\ell := (c_{1m}^\ell, c_{2m}^\ell, \dots, c_{N_\ell m}^\ell)^T$ the column vector which collects the coefficients used above. From (6) it follows that

$$M_{ij} = \sum_{\ell=\max\{\ell_i,\ell_j\}}^{n-1} (\mathbf{c}_i^\ell)^T M_\ell \mathbf{c}_j^\ell.$$

To assemble the global matrix, we define the matrix $C_\ell := [\mathbf{c}_1^\ell \mathbf{c}_2^\ell \dots \mathbf{c}_{N_{0:\ell}^A}^\ell]$, which collects the coefficients for basis change from hierarchical basis functions up to level ℓ to the basis $\mathcal{B}_\ell$. Then, the equation becomes

$$M = \sum_{\ell=0}^{n-1} [C_\ell \; \mathbf{0}]^T \, M_\ell \, [C_\ell \; \mathbf{0}]. \tag{7}$$

Notice that C_ℓ is a rectangular matrix of size $N_\ell \times N_{0:\ell}^A$, where we recall that $N_{0:\ell}^A = \sum_{k=0}^{\ell} N_k^A$ is the number of active functions up to level ℓ. The zero matrices in (7), of size $N_\ell \times (N - N_{0:\ell}^A)$, are added to have all the matrices in the sum of size $N \times N$. In practice, one can compute $C_\ell^T M_\ell C_\ell$ and add the resulting square matrix, of size $N_{0:\ell}^A \times N_{0:\ell}^A$, into the corresponding entries of M. The same procedure given in (7) can be used for the assembly of different matrices and vectors, in the latter the matrix multiplication is only done on the left.

The computation of the matrices C_ℓ can be done through the following recursive algorithm:

1. $C_0 = J_0$,
2. $C_{\ell+1} = [C_\ell^{\ell+1} C_\ell, \, J_{\ell+1}]$, for $\ell = 0, \dots, n-2$,

where in the first block of the matrix $C_{\ell+1}$ we compute, for $0 \le k < \ell+1$, the coefficients relative to active functions of level k of the matrices $C_k^{\ell+1}$ as in (3), and in the second block, $J_{\ell+1}$ denotes the inclusion of active functions of level $\ell+1$ into the whole tensor product basis of the same level. Notice that J_ℓ is a rectangular matrix of size $N_\ell \times N_\ell^A$, and in each column all entries are equal to zero with the exception of only one which is equal to one.

This approach of assembly works identically for THB-splines, because as already mentioned in §2.4, the restriction of a truncated function to an active element of level k is not affected by finer functions. The only difference is that, in the recursive algortihm, the matrices $C_\ell^{\ell+1}$ must be replaced by $C_\ell^{\ell+1,\tau}$.

Remark 2. *For the sake of clarity we have used the whole matrix C_ℓ, but for computational efficiency C_ℓ can be restricted to the rows of functions that do not vanish on active and deactivated elements, reducing memory consumption.*

Remark 3. *In our current Octave implementation the matrices C_ℓ (and also matrix K in §4.3) are computed explicitly and stored in memory, but the storage can be avoided by computing the coefficients on the fly whenever needed. For instance, in [12] the global matrices C_ℓ and M_ℓ are never assembled, and instead the same procedure is applied using elementary matrices, that are submatrices of C_ℓ. On the one hand, the use of a global matrix avoids to repeat computations, since many coefficients of the matrices are repeated between elements, and it does not require to compute the connectivity array, that is, the set of active functions in $\mathcal{H}$ that do not vanish on each element. On the other hand, global matrices increase the memory cost, while the coefficients and the connectivity array may be computed efficiently in compiled languages such as Fortran or*

C/C++, see for instance [27] where the authors take advantage of the uniform knot vector of Catmull-Clark subdivision for an efficient computation of the coefficients between levels. Another interesting approach is the one proposed by Bressan and Mokriš [28], that partition the subdomains Ω_ℓ into smaller regions where one can exploit the tensor product structure of the functions.

3.3. Rational hierarchical splines

The hierarchical construction can be generalized without difficulty to spaces of NURBS, considering that we have a nested sequence of NURBS spaces associated to the spline spaces $\{\mathcal{S}_\ell\}_{\ell\in\mathbb{N}_0}$. We recall that the rational basis functions of level ℓ are given by

$$R_{i,\ell} = \frac{w_{i,\ell}\beta_{i,\ell}}{w},$$

where we assume that $w \in \mathcal{S}_0$, and $w_{i,\ell}$ is the ith coefficient for writing w as a linear combination of the B-splines in $\mathcal{B}_\ell$, i.e., $w = \sum_{j=1}^{N_\ell} w_{j,\ell}\beta_{j,\ell}$. Since the weight w does not change during refinement, we can apply the two-scale relation (2) to obtain a similar relation between NURBS spaces (see also [7, 9, 29])

$$R_{i,\ell} = \frac{w_{i,\ell}\sum_{k=1}^{N_{\ell+1}} c_{k,\ell+1}(\beta_{i,\ell})\,\beta_{k,\ell+1}}{w} = \sum_{k=1}^{N_{\ell+1}} c_{k,\ell+1}(R_{i,\ell})R_{k,\ell+1},$$

with $c_{k,\ell+1}(R_{i,\ell}) = c_{k,\ell+1}(\beta_{i,\ell})\,w_{i,\ell}/w_{k,\ell+1}$. Then, one can apply the algorithm of the hierarchical construction to the NURBS spaces. For a given sequence of subdomains, the indices of the active functions in $\mathcal{H}$ do not differ between the rational and non-rational spaces. For the evaluation of basis functions, one should modify the matrix $C_\ell^{\ell+1}$ to consider the coefficients $c_{k,\ell+1}(R_{i,\ell})$ instead of $c_{k,\ell+1}(\beta_{i,\ell})$.

3.4. Multiple patches

Discretizations on multiple patch domains allow to treat complex geometries that cannot be represented as the image of the unit domain $\widehat{\Omega}$. The physical domain is given as the union of several closed patches, $\Omega = \bigcup_i \Omega^i$, defined through parametrizations $\mathbf{F}_i : \widehat{\Omega} \longrightarrow \Omega^i$. Defining a spline space on each subdomain Ω^i, and assuming that the knot vectors (the mesh) and the control points (the parametrization) coincide at the interface, one can construct spaces in the multipatch domain with global C^0 continuity, see [2, Section 3.5] and [3, Section 3.2].

To define hierarchical spaces in multipatch domains we follow [30]. First we define a globally continuous spline space $\mathcal{S}_0$ in the multipatch domain, using tensor product splines on each patch. Then, applying uniform refinement to all the patches, we define a multipatch space $\mathcal{S}_\ell$ for each level ℓ. Finally, selecting the subdomains Ω_ℓ as union of cells in the multipatch domain, we apply the hierarchical algorithm to this sequence of spaces, taking into account that functions on the interfaces are supported in more than one patch. We remark that the two-scale relation remains valid for the spaces on multipatch domains.

This approach is different from the one in [13], where a hierarchical space is constructed on each patch, and then the hierarchical spaces must be glued with C^0 continuity. The advantage of our approach is that the algorithms in the following sections work without any modification.

3.5. Boundary spaces

In isogeometric analysis it is common to construct the tensor product spline spaces $\{\mathcal{S}_\ell\}_{\ell\in\mathbb{N}_0}$ from open knot vectors. In this case it is possible to define, from the knot vectors and control points corresponding to each boundary side, the restriction to the boundary of the parametrization $\mathbf{F}$, and also the restriction to the boundary of the space $\mathcal{S}_\ell$ as a $(d-1)$-dimensional tensor product space (see for instance [17]). This automatically defines a Cartesian mesh on each boundary side.

The definition of boundary spaces is analogous for hierarchical splines: the restriction to the boundary of the hierarchical space $\operatorname{span}\mathcal{H}$ is a $(d-1)$-dimensional hierarchical space, and the hierarchical mesh in $\widehat{\Omega}$ automatically gives a hierarchical mesh on each boundary side. Notice that the number of levels of the boundary mesh and space is always less or equal than the number of levels of $\mathcal{H}$.

4. Refinement and coarsening of hierarchical splines

In order to develop adaptive methods based on hierarchical splines it is necessary to define suitable refinement and coarsening procedures. In this section we introduce such procedures for hierarchical splines, either marking by elements or by functions. We remark that our goal is not to introduce or analyse a particular adaptive strategy, but to present rigorous and robust procedures that can be used independently of the selected strategy. Indeed, we show that, unlike in previous works, our refinement and coarsening procedures are inverse of each other, in the sense that a refinement step can be reverted with a single coarsening step, and vice versa. The procedures that we introduce define a new hierarchy of subdomains, hence they work both for the standard and the simplified hierarchical basis, and they are valid for the truncated basis as well. We end this section showing how to compute a refinement matrix, to express a field in a coarse basis in terms of the refined basis.

4.1. Definition of a refined/coarsened hierarchical mesh and space

During an adaptive procedure, once a current hierarchical mesh $\mathcal{Q}$ and its corresponding hierarchical space $\operatorname{span}\mathcal{H}$ are given, we have to decide how to *refine* and/or *coarsen* the mesh, in order to get the new adapted space. Since the mesh is determined by the hierarchy of subdomains (4), it is necessary to generate a new sequence of subdomains.

Definition 4. *Let $\mathbf{\Omega}_n := \{\Omega_0, \Omega_1, \dots, \Omega_n\}$ and $\mathbf{\Omega}^*_{n^*} := \{\Omega^*_0, \Omega^*_1, \dots, \Omega^*_{n^*}\}$ be hierarchies of subdomains of $\widehat{\Omega}$ of depth n and n^*, respectively, where $n^* \geq n$. We say that $\mathbf{\Omega}^*_{n^*}$ is an* enlargement *of $\mathbf{\Omega}_n$, or equivalently, that $\mathbf{\Omega}_n$ is a* shrinking *of $\mathbf{\Omega}^*_{n^*}$, if*

$$\Omega_\ell \subset \Omega^*_\ell, \qquad \ell = 1, 2, \dots, n.$$

Let $\mathcal{H}$ and $\mathcal{Q}$ be a hierarchical B-spline basis and the corresponding hierarchical mesh, respectively, associated to the hierarchy of subdomains of depth n, $\mathbf{\Omega}_n := \{\Omega_0, \Omega_1, \dots, \Omega_n\}$. For $\mathbf{\Omega}^*_{n^*}$ an enlargement of $\mathbf{\Omega}_n$, we denote by $\mathcal{H}^*$ the associated *refined hierarchical basis* (see either Definition 2 or Definition 3), and by $\mathcal{Q}^*$ the corresponding *refined hierarchical mesh.* It holds that the enlargement of $\mathbf{\Omega}_n$ gives rise to a new enriched hierarchical space, in the sense that

$$\operatorname{span}\mathcal{H} \subset \operatorname{span}\mathcal{H}^*.$$

This result has been proved in [20, Proposition 6] for the standard hierarchical basis, and in [18, Theorem 5.9] for the simplified hierarchical basis.

4.1.1. Construction of an enlargement of the subdomains for refinement

In order to build an enlargement of the subdomains it is necessary to select the regions of the domain which require better approximation, following a marking strategy based, for instance, on some a posteriori error indicators. These regions are given as a collection of active cells to be refined, $\mathcal{M}^e \subset \mathcal{Q}$, and we denote by $\mathcal{M}^e_\ell = \mathcal{M}^e \cup \mathcal{Q}_\ell$ the marked cells of level ℓ, for $\ell = 0, 1, \dots, n-1$. With these selected elements, the enlargement $\mathbf{\Omega}^*_{n^*} := \{\Omega^*_0, \Omega^*_1, \dots, \Omega^*_{n^*}\}$ of depth (at most) $n+1$ is defined by

$$\begin{cases} \Omega^*_0 & := & \Omega_0, & \\ \Omega^*_\ell & := & \Omega_\ell \cup \left\{ Q \mid Q \in \mathcal{M}^e_{\ell-1} \right\}, & \ell = 1, 2, \dots, n, \\ \Omega^*_{n+1} & := & \emptyset, & \text{if } \Omega^*_n \neq \emptyset, \end{cases} \tag{8}$$

i.e., Ω^*_ℓ is obtained from Ω_ℓ by adding the marked cells of level $\ell - 1$. Notice that $\mathcal{M}^e \subset \mathcal{Q} \setminus \mathcal{Q}^*$, i.e., all the marked cells have been refined. We denote this refinement operation as

$$\mathbf{\Omega}_n \xrightarrow{\text{refine}(\mathcal{M}^e)} \mathbf{\Omega}^*_{n^*}. \tag{9}$$

The choice of active cells to be refined can be done either by directly marking active cells in $\mathcal{Q}$, as it is usually done in adaptive finite elements, or selecting to refine a set of active basis functions in $\mathcal{H}$, which is equivalent to refining their support [24]. More precisely, we consider the two following ways of refinement:

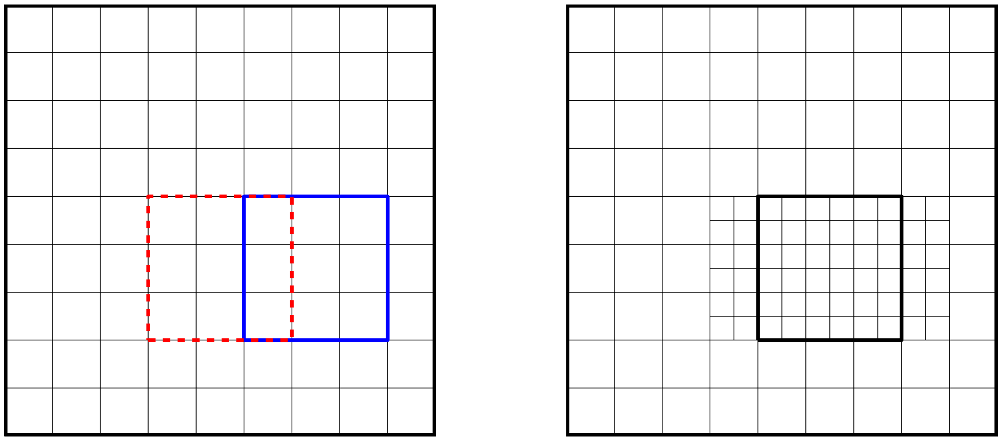


Figure 4: We consider biquadratic B-splines of maximum smoothness and a one-level mesh $\mathcal{Q}$ (i.e. $\Omega_0 = \widehat{\Omega}$ and $\Omega_1 = \emptyset$). The set of marked functions $\mathcal{M}_0^f$ consists of the two highlighted B-splines of level 0 (left). The refined mesh $\mathcal{Q}^*$ is shown on the right. Notice that the highlighted B-spline of level 0 is not in $\mathcal{H}^*$, i.e., it was deactivated.

- **Marking active cells:** the set of *marked* active cells $\mathcal{M}^e \subset \mathcal{Q}$ is directly given.
- **Marking basis functions:** a set of *marked* active basis functions $\mathcal{M}^f \subset \mathcal{H}$ is given. Let $\mathcal{M}_\ell^f := \mathcal{M}^f \cap \mathcal{B}_\ell$ be the marked functions of level ℓ, for $\ell = 0, 1, \ldots, n-1$. We then define $\mathcal{M}^e := \bigcup_{\ell=0}^{n-1} \mathcal{M}_\ell^e$, with
$$\mathcal{M}_\ell^e := \{Q \in \mathcal{Q}_\ell \cap \mathcal{Q} \mid \exists \beta \in \mathcal{M}_\ell^f \text{ such that } Q \subset \operatorname{supp} \beta\}, \tag{10}$$
that is, for each marked function we refine the active cells of its level contained in its support. In this case, as an alternative to (9), we write
$$\mathbf{\Omega}_n \xrightarrow{\text{refine}(\mathcal{M}^f)} \mathbf{\Omega}_{n^*}^*.$$

Once the enlargement $\mathbf{\Omega}_{n^*}^*$ has been defined, together with the sequence of spaces (1), the hierarchical basis $\mathcal{H}^*$ can be determined. Notice that when marking basis functions it holds $\mathcal{M}^f \subset \mathcal{H} \setminus \mathcal{H}^*$, i.e., all the marked basis functions have been deactivated, because their support has been refined. Moreover, it may happen that a function that has not been marked must be deactivated, because its support is contained in the union of the supports of other functions, as it happens in Figure 4.

4.1.2. Construction of a shrinking of the subdomains for coarsening

We introduce here a new methodology to construct the subdomains for coarsening from a selection of marked cells or marked functions to be reactivated. Compared to other algorithms existing in the literature, our definition of coarsening is less agressive than the ones in [13] and [12], because we only allow to reactivate cells such that their children have not been further refined (see (11)). Instead, in [13] they allow to reactivate the parent of an active cell without giving any restriction, and in [12] when marking a function to unrefine all the cells of its level contained in its support are reactivated, also without restrictions. Being more restrictive permits us to see coarsening as an inverse of refinement, as we will explain in detail in §4.2. A similar idea was proposed in [21], but their approach is more restrictive than ours, because they only allow to reactivate functions such that all their children are active (compare with (14)).

We start defining, for each level $\ell = 0, \ldots, n-2$, the set of elements admissible for reactivation as
$$\mathcal{D}_\ell^e = \{Q \in \mathcal{Q}_\ell \mid Q \subset \Omega_{\ell+1} \wedge Q \cap \operatorname{int}(\Omega_{\ell+2}) = \emptyset\}, \tag{11}$$
formed by deactivated elements such that none of their children has been deactivated. An example is shown in Figure 5: the two blue elements are admissible for reactivation, while the two red elements are not, because

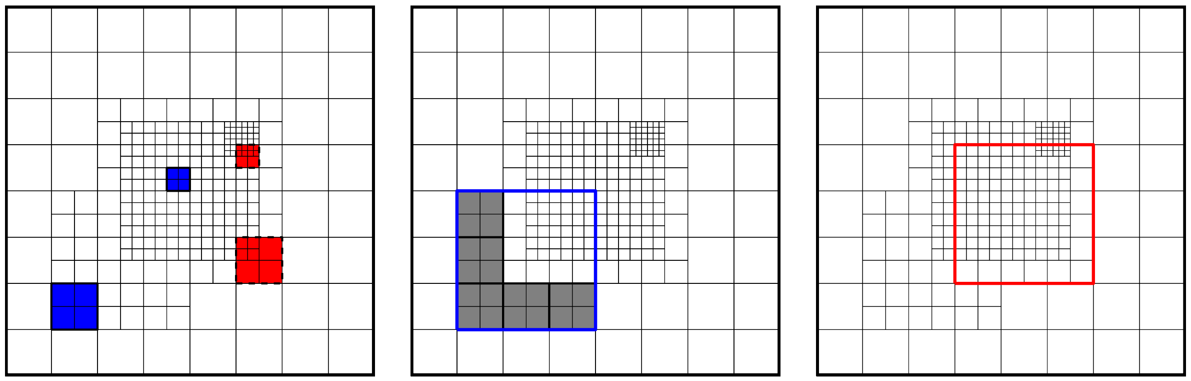


Figure 5: Examples of elements and functions admissible for reactivation. In the first figure, the two blue elements (solid lines) are admissible for reactivation, while the two red ones (dashed lines) are not. Considering biquadratic B-splines of maximum smoothness, the basis function of level 0 highlighted in the middle figure is admissible for reactivation, because there are five elements (in grey) of its level in its support that belong to $\mathcal{D}_0^e$. The highlighted function in the right figure is not admissible for reactivation, as none of the elements of its same level and in its support belongs to $\mathcal{D}_0^e$.

some of their children have been refined. To build a shrinking of the subdomains the region to be coarsened is represented by a subset of these elements $\mathcal{M}^e := \cup_{\ell=0}^{n-2} \mathcal{M}_\ell^e$, where $\mathcal{M}_\ell^e \subset \mathcal{D}_\ell^e$. With these selected elements, we define the shrinking $\mathbf{\Omega}_{n^-}^- := \{\Omega_0^-, \Omega_1^-, \dots, \Omega_{n^-}^-\}$ of depth $n^- \leq n$, by

$$\begin{cases} \Omega_0^- & := & \Omega_0, & \\ \Omega_\ell^- & := & \Omega_\ell \setminus \{Q \mid Q \in \mathcal{M}_{\ell-1}^e\}, & \ell = 1, 2, \dots, n-1, \\ \Omega_n^- & := & \emptyset, & \text{if } \Omega_{n-1}^- \neq \emptyset, \end{cases} \tag{12}$$

i.e., Ω_ℓ^- is obtained from Ω_ℓ by removing the cells selected for unrefinement (to be reactivated) of level $\ell - 1$. Notice that, denoting by $\mathcal{Q}^-$ the hierarchical mesh corresponding to the subdomains $\mathbf{\Omega}_{n^-}^-$, it holds that $\mathcal{M}^e \subset \mathcal{Q}^- \setminus \mathcal{Q}$, i.e., all cells selected for unrefinement have been reactivated. We denote this coarsening operation as

$$\mathbf{\Omega}_n \xrightarrow{\text{coarsen}(\mathcal{M}^e)} \mathbf{\Omega}_{n^-}^-. \tag{13}$$

Similar to what we have seen for the refinement, the choice of cells to be reactivated for coarsening can be done either by marking elements or by marking functions. For this reason, we need to define the set of functions admissible for reactivation as

$$\mathcal{D}_\ell^f = \{\beta \in \mathcal{F}_\ell^D \mid \exists Q \in \mathcal{D}_\ell^e \text{ such that } Q \subset \operatorname{supp} \beta\}, \tag{14}$$

the set of deactivated functions that have an element in $\mathcal{D}_\ell^e$ within its support. Two examples are given in Figure 5, one of a function admissible for reactivation and one for a function that is not. In particular, any function in $\mathcal{D}_\ell^f$ has at least one active child, but not all of them are necessarily active. However, we remark that having an active child is not a sufficient condition to be in $\mathcal{D}_\ell^f$.

We then consider the two following ways of coarsening:

- **Marking cells to reactivate:** a set of deactivated cells $\mathcal{M}^e := \cup_{\ell=0}^{n-2} \mathcal{M}_\ell^e$, with $\mathcal{M}_\ell^e \subset \mathcal{D}_\ell^e$, must be given.
- **Marking functions to reactivate:** a set of deactivated functions $\mathcal{M}^f := \cup_{\ell=0}^{n-2} \mathcal{M}_\ell^f$, with $\mathcal{M}_\ell^f \subset \mathcal{D}_\ell^f$, must be given. Using these we define the set of cells to be reactivated as

$$\begin{aligned} \mathcal{M}_\ell^e := \{Q \in \mathcal{D}_\ell^e \mid & \exists \beta \in \mathcal{M}_\ell^f \text{ such that } Q \subset \operatorname{supp} \beta, \\ & \text{and } \forall \beta' \in \mathcal{F}_\ell^D \setminus \mathcal{M}_\ell^f, Q \not\subset \operatorname{supp} \beta'\}, \end{aligned} \tag{15}$$

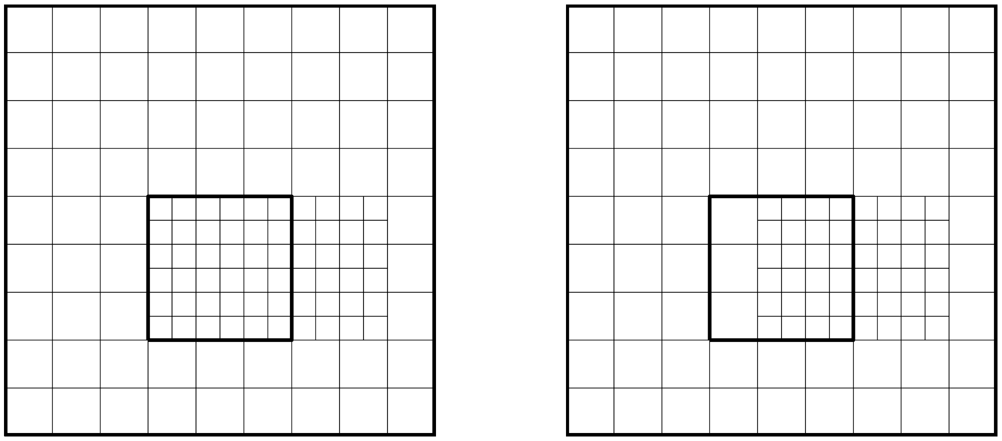


Figure 6: An example of coarsening by marking functions. Considering biquadratic B-splines of maximum smoothness, in the mesh on the left, there exist three functions of level 0 that are deactivated. Marking the highlighted function to be reactivated generates the mesh on the right figure, where only three elements are reactivated. The other elements in the support of the function remain as they are, because they are shared by the other two functions, that have not been marked.

that is, we reactivate the cells in the support of functions in $\mathcal{M}^f$ whenever this coarsening does not reactivate functions that are not in $\mathcal{M}^f$. In other words, functions that have not been marked should remain deactivated. An example is shown in Figure 6: when marking the highlighted function, only three elements in its support are reactivated, because the other elements are shared by two other functions that are not in $\mathcal{M}^f$. As we did for refinement, as an alternative to (13) we write

$$\mathbf{\Omega}_n \xrightarrow{\text{coarsen}(\mathcal{M}^f)} \mathbf{\Omega}^-_{n^-}.$$

Once the shrinking $\mathbf{\Omega}^-_{n^-}$ has been defined, the corresponding hierarchical basis $\mathcal{H}^-$ can be constructed. When marking basis functions, only functions in $\mathcal{M}^f_\ell$ with a cell in $\mathcal{M}^e_\ell$ within their supports are reactivated, which means that not necessarily all marked functions are reactivated. For instance, in the example of Figure 4 (right) marking only the highlighted function to be reactivated would not cause any coarsening, because all the cells in its support are contained in the support of at least one of the two highlighted functions in Figure 4 (left), that have not been marked.

Notice that the list of entities to be reactivated may be built from some a posteriori error indicators, that usually work on *active* elements or functions. For instance, when marking cells one can choose to reactivate a cell in $\mathcal{D}^e_\ell$ if all its children are marked, or if at least one of them is marked. We do not focus on a particular strategy here, but present how to define the shrinking assuming that a list of entities to be reactivated has been defined. Some examples of how to choose the entities to reactivate are tested in §7.2.2.

Remark 4. *As we said above, the algorithms in [21] are more restrictive than ours, because they only allow to reactivate functions such that all their children are active, and analogously, they only allow to refine functions such that all their parents have been deactivated. Although this restriction is relaxed in the companion paper [25], the algorithms in those papers do not guarantee the linear independence of the active functions for high order splines. A partial remedy was introduced in [31], where a check of the support and the number of represented children is performed during refinement, to avoid linear dependencies. However, the check they include in the coarsening algorithms is not sufficient, and may still lead to linear dependencies.*

4.2. Coarsening as the inverse of refinement

The coarsening as we defined it can be seen as the inverse operation of refinement. When marking cells, it is trivial to see that if we apply a refinement step with marked cells $\mathcal{M}^e = \cup_{\ell=0}^{n-1} \mathcal{M}^e_\ell$ and then a coarsening step with the same set of cells to be reactivated, we obtain the original mesh (see Proposition 1 below). When marking basis functions, instead, after one refinement step we have to mark for reactivation not only the functions that have been marked for refinement, but all those that have been deactivated (see Proposition 2 below). For instance, in the example of Figure 4 we marked two functions for refinement, but to recover the original mesh it is necessary to mark the three deactivated functions to be reactivated.

Proposition 1. *Let $\mathbf{\Omega}_n$ be a hierarchy of subdomains of depth n and $\mathcal{Q}$ be the corresponding hierarchical mesh. Let us define the sequence of refinement and coarsening:*

$$\mathbf{\Omega}_n \xrightarrow{\text{refine}(\mathcal{M}^e)} \mathbf{\Omega}^+_{n^+} \xrightarrow{\text{coarsen}(\mathcal{M}^e)} \mathbf{\Omega}^-_{n^-},$$

where $\mathcal{M}^e \subset \mathcal{Q}$. Then $n^- = n$, and $\Omega^-_\ell = \Omega_\ell$, for $\ell = 0, \dots, n$.

Analogously, let us define the sequence of coarsening and refinement:

$$\mathbf{\Omega}_n \xrightarrow{\text{coarsen}(\mathcal{M}^e)} \mathbf{\Omega}^-_{n^-} \xrightarrow{\text{refine}(\mathcal{M}^e)} \mathbf{\Omega}^+_{n^+},$$

where $\mathcal{M}^e = \cup_{\ell=0}^{n-2} \mathcal{M}^e_\ell$, with $\mathcal{M}^e_\ell \subset \mathcal{D}^e_\ell$. Then $n^+ = n$, and $\Omega^+_\ell = \Omega_\ell$, for $\ell = 0, \dots, n$.

Proof. The proof is trivial from the definitions in (8) and (12). □

Proposition 2. *Let $\mathbf{\Omega}_n$ be a hierarchy of subdomains of depth n where Ω_ℓ is the union of supports of B-splines of level $\ell-1$, for $\ell = 1, \dots, n-1$, and let us define the sequence of refinement and coarsening marking by functions:*

$$\mathbf{\Omega}_n \xrightarrow{\text{refine}(\mathcal{M}^f)} \mathbf{\Omega}^+_{n^+} \xrightarrow{\text{coarsen}(\widehat{\mathcal{M}}^f)} \mathbf{\Omega}^-_{n^-}, \tag{16}$$

where $\mathcal{M}^f := \bigcup_{\ell=0}^{n-1} \mathcal{M}^f_\ell$, with $\mathcal{M}^f_\ell \subset \mathcal{B}_\ell \cap \mathcal{H}$, and $\widehat{\mathcal{M}}^f := \bigcup_{\ell=0}^{n-1} \widehat{\mathcal{M}}^f_\ell$, where $\widehat{\mathcal{M}}^f_\ell := \{\beta \in \mathcal{B}_\ell \cap \mathcal{H} \mid \operatorname{supp}\beta \subset \cup_{\beta' \in \mathcal{M}^f_\ell} \operatorname{supp}\beta'\}$ are the functions of level ℓ deactivated in the refinement step. Then $n^- = n$, and $\Omega^-_\ell = \Omega_\ell$, for $\ell = 0, \dots, n$.

Analogously, let the sequence of coarsening and refinement:

$$\mathbf{\Omega}_n \xrightarrow{\text{coarsen}(\mathcal{M}^f)} \mathbf{\Omega}^-_{n^-} \xrightarrow{\text{refine}(\check{\mathcal{M}}^f)} \mathbf{\Omega}^+_{n^+}, \tag{17}$$

where $\mathcal{M}^f := \bigcup_{\ell=0}^{n-2} \mathcal{M}^f_\ell$ is a subset of functions admissible for reactivation, and $\check{\mathcal{M}}^f := \bigcup_{\ell=0}^{n-2} \check{\mathcal{M}}^f_\ell$, where $\check{\mathcal{M}}^f_\ell \subset \mathcal{M}^f_\ell$ is the set of functions of level ℓ that have been actually reactivated. Then $n^+ = n$, and $\Omega^+_\ell = \Omega_\ell$, for $\ell = 0, \dots, n$.

Proof. We start proving the result for the sequence (16). First, notice that $\widehat{\mathcal{M}}^f$ is a subset of functions admissible for reactivation as defined in §4.1.2. Let $\mathcal{M}^e := \cup_{\ell=0}^{n-1} \mathcal{M}^e_\ell$ the set of elements to refine obtained from $\mathcal{M}^f$ as in (10), and $\widehat{\mathcal{M}}^e := \cup_{\ell=0}^{n-1} \widehat{\mathcal{M}}^e_\ell$ the set of elements to reactivate obtained from $\widehat{\mathcal{M}}^f$ (see (15)), i.e.,

$$\widehat{\mathcal{M}}^e_\ell := \{Q \in \mathcal{D}^{e,+}_\ell \mid \exists \beta \in \widehat{\mathcal{M}}^f_\ell \text{ s.t. } Q \subset \operatorname{supp}\beta, \text{ and } \forall \beta' \in \mathcal{F}^{D,+}_\ell \setminus \widehat{\mathcal{M}}^f_\ell,\, Q \not\subset \operatorname{supp}\beta'\}. \tag{18}$$

Taking into account Proposition 1, we only need to prove that $\mathcal{M}^e_\ell = \widehat{\mathcal{M}}^e_\ell$ for $\ell = 0, \dots, n-1$.

Let $Q \in \mathcal{M}^e_\ell$. Since Q is a marked active element in the initial mesh $\mathcal{Q}$, we have that Q is admissible for reactivation in $\mathcal{Q}^+$, that is, $Q \in \mathcal{D}^{e,+}_\ell$. Moreover, there exists $\beta \in \mathcal{M}^f_\ell$ such that $Q \subset \operatorname{supp}\beta$, and by definition it also holds that $\beta \in \widehat{\mathcal{M}}^f_\ell$. Finally, if $\beta' \in \mathcal{F}^{D,+}_\ell \setminus \widehat{\mathcal{M}}^f_\ell$, then $\beta' \in \mathcal{F}^D_\ell$, that is, β is a deactivated function in the initial mesh $\mathcal{Q}$, which in turn implies that all elements within its support are deactivated in $\mathcal{Q}$. Since Q is active in $\mathcal{Q}$, $Q \not\subset \operatorname{supp}\beta'$. Therefore, $Q \in \widehat{\mathcal{M}}^e_\ell$.

Now let $Q \in \widehat{\mathcal{M}}^e_\ell$. To prove that $Q \in \mathcal{M}^e_\ell$ we have to see that Q is active in the initial mesh $\mathcal{Q}$, and that $Q \subset \operatorname{supp}\beta$ for some $\beta \in \mathcal{M}^f_\ell$. Let us suppose that Q was deactivated in $\mathcal{Q}$, that is $Q \subset \Omega_{\ell+1}$. Since $\Omega_{\ell+1}$ is a union of supports of functions in $\mathcal{B}_\ell$, there exists $\beta' \in \mathcal{B}_\ell$ such that $Q \subset \operatorname{supp}\beta' \subset \Omega_{\ell+1}$. Then, $\beta' \in \mathcal{F}^D_\ell$, and clearly $\beta' \in \mathcal{F}_\ell^{D,+} \setminus \widehat{\mathcal{M}}_\ell^f$, which contradicts (18). Therefore, Q is active in the initial mesh $\mathcal{Q}$. Moreover, from (18) it follows that Q is deactivated in $\mathcal{Q}^+$, which means that $Q \subset \operatorname{supp} \beta$ for some $\beta \in \mathcal{M}_\ell^f$. Thus, $Q \in \mathcal{M}_\ell^e$. This proves the result for the sequence (16).

The proof for sequence (17) is similar. We have to prove that $\mathcal{M}_\ell^e = \check{\mathcal{M}}_\ell^e$, for $\ell = 0, 1, \ldots, n-2$, where $\mathcal{M}_\ell^e$ are the elements to reactivate obtained from $\mathcal{M}^f$ as in (15), and $\check{\mathcal{M}}_\ell^e$ are the elements to refine in $\mathcal{Q}^-$ obtained from $\check{\mathcal{M}}_\ell^f$ (see (10)), i.e.,

$$\check{\mathcal{M}}_\ell^e := \{Q \in \mathcal{Q}_\ell \cap \mathcal{Q}^- \mid \exists \beta \in \check{\mathcal{M}}_\ell^f \text{ such that } Q \subset \operatorname{supp} \beta\}. \tag{19}$$

Let $Q \in \mathcal{M}_\ell^e$. From (15) it follows that $Q \subset \operatorname{supp} \beta$ for some $\beta \in \mathcal{M}_\ell^f$. Since all elements in $\mathcal{M}_\ell^e$ have been reactivated, we have that Q is an active element in $\mathcal{Q}^-$, which in turn implies that β has been actually reactivated, that is, $\beta \in \check{\mathcal{M}}_\ell^f$. Hence, taking into account (19), $Q \in \check{\mathcal{M}}_\ell^e$.

Now let $Q \in \check{\mathcal{M}}_\ell^e$, by (19) we have $Q \subset \operatorname{supp} \beta$ for some $\beta \in \check{\mathcal{M}}_\ell^f \subset \mathcal{M}_\ell^f$. Since $\beta \in \mathcal{F}_\ell^D$, Q is a deactivated element in $\mathcal{Q}$. Moreover, $Q \in \mathcal{D}_\ell^e$, because Q is active in $\mathcal{Q}^-$. In addition, taking into account that the nonmarked deactivated functions remain deactivated, that is, $\mathcal{F}_\ell^D \setminus \mathcal{M}_\ell^f \subset \mathcal{F}_\ell^{D,-}$, we have that $Q \not\subset \operatorname{supp} \beta'$ for any $\beta' \in \mathcal{F}_\ell^D \setminus \mathcal{M}_\ell^f$. Thus, all the conditions of (15) are satisfied, which implies $Q \in \mathcal{M}_\ell^e$, and the result is proved. $\square$

4.3. Refinement matrix between hierarchical bases

When applying a refinement step, it may be necessary to express a field computed in the coarse space in terms of the new basis $\mathcal{H}^*$, for instance to compute the new control points of the parametrization $\mathbf{F}$, to get the coefficients of the partition of unity, or in a time discretization with adaptivity to pass the solution of previous time steps to the current mesh. Due to the nestedness of the spaces this operation can be done exactly as a matrix vector multiplication, in the form

$$\mathbf{u}^* = K\mathbf{u},$$

where $\mathbf{u}$ and $\mathbf{u}^*$ are the vectors collecting the coefficients in terms of the bases $\mathcal{H}$ and $\mathcal{H}^*$, respectively, and each column of the matrix K gives the coefficients to express a basis function in $\mathcal{H}$ in terms of the basis $\mathcal{H}^*$.

As already mentioned in Remark 4, in [21] the authors only allow to refine functions such that all their parents are already deactivated, and therefore each refined (deactivated) basis function only needs to pass the information to its children, using the two-scale relation (2). In our case we do not impose that restriction, therefore a refined function must propagate the information through finer levels until all its active descendants are reached, using the coefficients in (3).

The computation of the coefficients of the matrix K was explained in [12, §3.4] for the case when the starting space is the tensor product space of level 0. We now explain the computation of the matrix K through a recursive algorithm in the general refinement situation between two nested hierarchical spaces, starting with the case of standard (non truncated) hierarchical splines. For simplicity, we assume that the ordering of the basis functions is the same as in §3.2.

By the refinement procedure, it always happens that $\mathcal{F}_\ell^A \subset (\mathcal{F}_\ell^{A^*} \cup \mathcal{F}_\ell^{D^*})$, i.e., any active function in the coarse basis is either active or deactivated in the fine basis, and in the latter case the information must be passed to the descendants. Let us define, similarly to the matrix J_ℓ in §3.2, the inclusion matrix of size $(N_\ell^{A^*} + N_\ell^{D^*}) \times N_\ell^A$, that we denote by $\widehat{J}_\ell$, to pass from functions in $\mathcal{F}_\ell^A$ to functions in $\mathcal{F}_\ell^{A^*} \cup \mathcal{F}_\ell^{D^*}$. Starting from $K_0 = \widehat{J}_0$, the matrix $K = K_{n^*-1}$ is computed as the last step in the following:

$$K_{\ell+1} = \begin{bmatrix} K_\ell^{A^*} & 0 \\ K_\ell^{\ell+1} K_\ell^{D^*} & \widehat{J}_{\ell+1} \end{bmatrix}, \qquad \text{for } \ell = 0, \ldots, n^* - 2,$$

where, in the first row we simply keep active functions, and the matrix $K_\ell^{A^*}$ is the restriction of K_ℓ to the rows of active functions in $\mathcal{H}^*$ up to level ℓ; and in the second row we pass the coefficients of deactivated functions to their children: the matrix $K_\ell^{D^*}$ is the submatrix of K_ℓ restricted to the rows corresponding to deactivated functions of level ℓ in $\mathcal{H}^*$, and the matrix $K_\ell^{\ell+1}$ is the submatrix of $C_\ell^{\ell+1}$ restricted to the rows of active and deactivated functions of level $\ell+1$ in $\mathcal{H}^*$, and the columns of deactivated functions of level ℓ in $\mathcal{H}^*$.

The computation of the refinement matrix in the case of THB-splines is slightly different, for two main reasons. First, a function that remains active may be (further) truncated, and part of its information must travel to its (new) truncating descendants, therefore we cannot use in the second row the restriction to deactivated functions; second, a function that is refined only needs to pass its information to new active descendants, since those that were already active collected the information coming from truncation in previous steps, which is done replacing the matrix $C_\ell^{\ell+1}$ with the matrix $C_\ell^{\ell+1,\tau}$ introduced in §2.4. Moreover, the computation of the coefficients for truncation may involve information passing through passive functions, like in the example of Figure 7: the coefficient to pass information from the basis function of level 0 (column index in K) to the highlighted function of level 2 (row index in K) requires the contribution from the highlighted passive function of level 1. Therefore we cannot restrict the computations to active and deactivated functions.

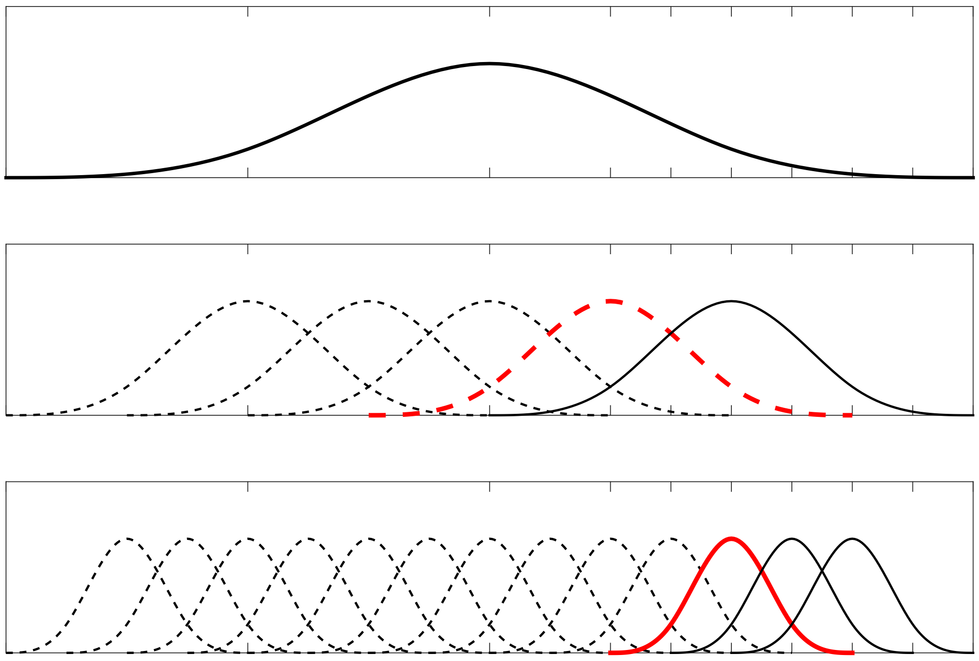


Figure 7: A cubic function of level 0, and its active (continuous) and passive (dashed) descendants of levels 1 and 2. When activating the highlighted function of level 2 (thick red line), the computation of the refinement matrix in the truncated basis requires passing through one passive function of level 1 (dashed thick red line).

To compute the matrix K for THB-splines, using now the matrices J_ℓ as in §3.2, we start from $K_0 = J_0$ and compute the matrices K_ℓ similarly to the non-truncated case:

$$K_{\ell+1} = \begin{bmatrix} K_\ell^{A^*} & 0 \\ C_\ell^{\ell+1,\tau} K_\ell^* & J_{\ell+1} \end{bmatrix}, \text{ for } \ell = 0, \dots, n^* - 2,$$

where $K_\ell^{A^*}$ is, as before, the restriction of K_ℓ to the rows of active functions up to level ℓ, and in the second row K_ℓ^* is the restriction of K_ℓ to rows of functions of level ℓ (active, deactivated or passive), which removes functions from previous levels, and the matrix $C_\ell^{\ell+1,\tau}$ is the one introduced in §2.4 constructed from the truncated hierarchical basis *before refinement*. The final step is to remove the non-active functions of the finest level, thus the refinement matrix is defined as

$$K = K^{A^*}_{n^*-1}.$$

Notice that, similar to Remark 2, the computation of the matrices J_ℓ and $C_\ell^{\ell+1,\tau}$ can be restricted to the submatrices corresponding to functions acting on active and deactivated elements, reducing the memory consumption.

Finally, we remark that after a coarsening step, a field in the original space cannot be represented exactly in the coarse space, and it is required to apply some approximation methods, such as the least-squares method [12] or a quasi-interpolant as in [18, 23]. We do not explain the details here, since the computations will depend on the chosen method.

5. Data structures for the implementation

We introduce in this section the main data structures and functions that are necessary for the implementation of adaptive isogeometric methods based on hierarchical splines, and that will be used in §6 to write the refinement and coarsening algorithms. We introduce two main data structures: one for the mesh, with the list of active elements, and one for the discrete space, with the list of active basis functions. These data structures are similar to those used in [12] in the case of uniform refinement between levels. As already mentioned in the introduction, our data structures are simpler and more intuitive than those in [13] and that the binary subdivision tree used in [14], and are still general enough to cover not only the different hierarchical bases defined in §2, but in general any hierarchical basis constructed under the setting in [20].

5.1. Data structures for the tensor product spaces

Our starting point is the existence of a code capable to perform an isogeometric discretization with a space of one single level, as those in §2.1. We assume that for any tensor product space, and therefore for each level ℓ, we have a *mesh* structure (or class) with the information of the Cartesian grid $\mathcal{Q}_\ell$, and a *space* structure (or class) with the information of the tensor product space $\mathcal{S}_\ell$. To solve a variational IGA problem with tensor product spaces, these structures are complemented with functions (or methods) to evaluate the basis functions of $\mathcal{B}_\ell$ and their derivatives at the quadrature points, to compute local elementary matrices, and to assemble the global matrix of the problem. Moreover, we also need the following functions, that work in the mesh and the space of one single level:

- **get_basis_functions**: for a given cell $Q \in \mathcal{Q}_\ell$ (or an array of cells), compute the indices of the basis functions in $\mathcal{B}_\ell$ that do not vanish in Q. In IGA software this is equivalent to the computation of the connectivity, which is used for assembling the matrix.
- **get_cells**: for a given basis function $\beta \in \mathcal{B}_\ell$ (or an array of functions), compute the cells in $\mathcal{Q}_\ell$ in which the function does not vanish.
- **get_neighbors**: for a given basis function $\beta \in \mathcal{B}_\ell$ (or an array of functions), compute the indices of basis functions in $\mathcal{B}_\ell$ such that their supports share at least one cell with the support of β. In IGA software, this information is used for computing the sparsity pattern of the matrix. A possible implementation of this function is as a composition of the previous two.

Notice that, since all these functions are for tensor product spaces on Cartesian grids, the computations can be done in the univariate case, and then generalized to the multivariate case by tensorization[1].

[1]Octave and Matlab users will find the functions `ind2sub`, `sub2ind` and `kron` extremely useful.

5.2. Data structure for the hierarchical mesh

The first structure (or class) must contain the information for the hierarchical mesh $\mathcal{Q}$. The essential information that is required to be stored in this structure is the following:

- The current number of levels.
- The Cartesian mesh structure for each level ℓ, with the information for $\mathcal{Q}_\ell$.
- The list of active cells for each level ℓ, that we denote by $\mathtt{E}_\ell^{\mathtt{A}}$ in the algorithms.
- The list of deactivated cells for each level ℓ, that we denote by $\mathtt{E}_\ell^{\mathtt{D}}$ in the algorithms.
- The mesh refinement to pass from level ℓ to $\ell + 1$ (dyadic, triadic...).

Moreover, we will also need functions to relate the information of two consecutive levels, that will depend on the choice of the spaces of the sequence (1). Usually this will be a simple dyadic h-refinement, but our algorithms are general enough to cover any other refinement, provided that the spline spaces are nested. The two functions that we will need for the mesh are:

- **get_children_of_cell**: for a given cell $Q \in \mathcal{Q}_\ell$ (or an array of cells), compute the indices of its children, that is, the cells $Q' \in \mathcal{Q}_{\ell+1}$ such that $Q' \subset Q$.
- **get_parent_of_cell**: for a given cell $Q \in \mathcal{Q}_{\ell+1}$ compute the index of its parent, that is, the unique cell $Q' \in \mathcal{Q}_\ell$ such that $Q \subset Q'$.

Remark 5. *The hierarchical mesh can be implemented in a tree structure, that becomes a simple quadtree or octree for dyadic h-refinement. In this structure the active elements are the leaves, and the other nodes of the tree are the deactivated elements. This kind of implementation has been already used in [9], for instance.*

5.3. Data structure for the hierarchical space

The second structure (or class) must contain all the information for the computation of the hierarchical basis. This is the list of necessary fields in the structure:

- The current number of levels.
- For each level ℓ, a space structure with the information for the tensor product basis $\mathcal{B}_\ell$.
- The list of active functions $\mathcal{F}_\ell^A$ for each level ℓ, written $\mathtt{F}_\ell^{\mathtt{A}}$ in the algorithms.
- The list of deactivated functions $\mathcal{F}_\ell^D$ for each level ℓ, written $\mathtt{F}_\ell^{\mathtt{D}}$ in the algorithms.
- The coefficients for the (univariate) two-scale relation between levels ℓ and $\ell + 1$.

As for the hierarchical mesh, we need some functions to relate basis functions of two consecutive levels:

- **get_children_of_function**: for a given basis function $\beta \in \mathcal{B}_\ell$ (or an array of functions), compute the indices of the children $\mathcal{C}(\beta)$, as defined in §2.1. Eventually, the function can also compute the coefficients of the two-scale relation (2). For h-refinement (respectively, p-refinement), these coefficients can be computed by a simple modification of knot insertion (degree elevation) algorithms.
- **get_parents_of_function**: for a given basis function $\beta \in \mathcal{B}_{\ell+1}$, compute the indices of its parents $\mathcal{P}(\beta)$.

We remark that the necessary fields in the structure are independent of the chosen basis, but the list of active and deactivated functions for the bases $\mathcal{H}$ and $\widetilde{\mathcal{H}}$ will differ, and at each refinement/coarsening step will be updated using different algorithms. The structure is also valid for truncated basis functions. In this case, there is no difference in the lists of indices of active and deactivated basis functions between the truncated and the non-truncated basis.

As already explained in §2.4, truncation can be incorporated into the two-scale relation by setting some rows of the matrix to zero. We note that this has to be done in the multivariate matrix, computed by Kronecker tensor product of the univariate ones, that we store in the structure.

Remark 6. *Unlike the hierarchical mesh, the hierarchical basis cannot be implemented as a standard tree structure, because a function of level $\ell+1$ can be a child of several functions of level ℓ. Moreover, the number of children differs between functions, and an active function would not necessarily be a leave of the tree, because one of its children could also be active, or even deactivated.*

6. Algorithms for refinement and coarsening

We now present the algorithms for the implementation of refinement and coarsening. Similar to the usage of cell-arrays in Octave/Matlab, in the algorithms we will write $\{\mathtt{A}_\ell\}_{\ell=0}^{n-1}$ for variables where the size varies from one level to another, and simply $\mathtt{A}_\ell$ when one single level is considered. To alleviate notation, the index interval in the cell-arrays will be specified only when it is different from 0 to $n-1$, that is, we will usually write $\{\mathtt{A}_\ell\}$. Also similar to Octave/Matlab, we will use brackets $[\mathtt{A},\mathtt{B}]$ to indicate that a function gives two (or more) output variables.

Both for refinement and coarsening, we assume that we are given a hierarchical mesh structure and a hierarchical space structure, that in the algorithms we denote MESH and SPACE, respectively. These contain the information explained in §5. Apart from these two structures, we get as an input a set of marked entities for each level, either cells or basis functions, that we denote by $\{\mathtt{MARKED}_\ell\}$. An important difference of our algorithms with respect to [12] is the way to update the lists of active and deactivated cells and functions. Our algorithms take advantage of the information in $\{\mathtt{MARKED}_\ell\}$ to update these lists locally, by simply adding or removing indices, while in [12] some of the lists are recomputed from scratch.

6.1. Refinement algorithms

Algorithm 1 summarizes the main steps to provide the refined hierarchical mesh and space, that is, the structures for the mesh $\mathcal{Q}^*$ and the basis $\mathcal{H}^*$. The algorithm is divided in two main parts: in the first part we compute the cells that have to be deactivated, and refine the hierarchical mesh, updating the sets of active and deactivated cells of each level; in the second part we compute the set of functions that have to be deactivated, and refine the hierarchical space, updating the sets of active and deactivated basis functions of each level. In the following paragraphs we explain each of these steps in detail.

We note that it would also be possible to refine the mesh and the space at the same time, as it is done for instance in [12, 13]. However, this approach of separating the mesh and the space can be more convenient in several situations. For example, in the mixed formulation for Stokes problem two discrete spaces, one for the velocity and one for the pressure, have to be refined simultaneously. In Algorithm 1 this would be simply done applying the second part of the algorithm to each discrete space.

Refinement of the hierarchical mesh. We first compute the set of cells to be deactivated, that we denote by $\{\mathtt{ME}_\ell\}$. This is done in Algorithm 10, that is detailed in Appendix A.

Once we have the set of cells to be deactivated, the refinement of the mesh described in Algorithm 2 is quite standard. First we check whether it is necessary to add a new level. Then, for each level, we remove marked cells from the list of active cells and add them to the list of deactivated cells, and add their children to the list of active cells of the next level. The algorithm gives as the output the new refined mesh, and also the list of cells that have been activated, that will be used to update the hierarchical basis.

Algorithm 1 refine: update MESH and SPACE when enlarging the current subdomains with the marked entities (either active cells or active basis functions) given in $\{\mathtt{MARKED}_\ell\}$

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
 1: if (Marking cells) then
 2:     $\{\mathtt{ME}_\ell\} \leftarrow \{\mathtt{MARKED}_\ell\}$
 3: else if (Marking functions) then
 4:     $\{\mathtt{ME}_\ell\} \leftarrow$ compute_cells_to_refine $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\})$
 5: end if
 6: $[\mathtt{MESH}, \{\mathtt{NE}_\ell\}_{\ell=0}^n] \leftarrow$ refine_hierarchical_mesh $(\mathtt{MESH}, \{\mathtt{ME}_\ell\})$

 7: if (Marking cells) then
 8:     $\{\mathtt{MF}_\ell\} \leftarrow$ functions_to_deactivate_from_cells $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\})$
 9: else if (Marking functions) then
10:     $\{\mathtt{MF}_\ell\} \leftarrow$ functions_to_deactivate_from_neighbors $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\})$
11: end if
12: $\mathtt{SPACE} \leftarrow$ refine_hierarchical_space $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}, \{\mathtt{NE}_\ell\}_{\ell=0}^n)$
    Output: $\mathtt{MESH}, \mathtt{SPACE}$
```

Algorithm 2 refine_hierarchical_mesh

```
    Input: $\mathtt{MESH}, \{\mathtt{ME}_\ell\}$
 1: if $\mathtt{ME}_{n-1} \neq \emptyset$ then
 2:     $\mathtt{MESH} \leftarrow$ add_empty_level
 3: end if
 4: $\mathtt{NE}_0 \leftarrow \emptyset$
 5: for $\ell = 0, \dots, n-1$ do
 6:     $\mathtt{E}^{\mathtt{A}}_\ell \leftarrow \mathtt{E}^{\mathtt{A}}_\ell \setminus \mathtt{ME}_\ell$
 7:     $\mathtt{E}^{\mathtt{D}}_\ell \leftarrow \mathtt{E}^{\mathtt{D}}_\ell \cup \mathtt{ME}_\ell$
 8:     $\mathtt{NE}_{\ell+1} \leftarrow$ get_children_of_cell $(\mathtt{ME}_\ell)$
 9:     $\mathtt{E}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{E}^{\mathtt{A}}_{\ell+1} \cup \mathtt{NE}_{\ell+1}$
10: end for
    Output: $\mathtt{MESH}, \{\mathtt{NE}_\ell\}_{\ell=0}^n$
```

Refinement of the space. Once the mesh information has been updated, we start with the refinement of the discrete space. As we did for the mesh, the first step is to collect the basis functions that have to be deactivated, that is, those functions whose support has no active cell of their same level in the refined mesh. This is done in Algorithm 11 when marking cells, and in Algorithm 12 when marking basis functions, both of them are detailed in Appendix A.

After selecting the functions that have to be deactivated, we can refine the hierarchical space, that is, we update the sets of active and deactivated basis functions for each level. The procedure is different depending on whether we work with the standard hierarchical space, or with the simplified hierarchical space, hence we explain them separately.

For the standard hierarchical space we use Algorithm 3. We start checking if we are refining elements of the finest level, to add an empty level to the space. Then for each level ℓ we remove the functions that have to be deactivated from the list of active functions, and add them to the list of deactivated ones. Next, for level $\ell + 1$ we collect the non-active functions such that their support intersects the new cells, which are candidate functions to be activated. In lines 9–14 we remove from this list those functions that are not completely supported in $\Omega^*_{\ell+1}$, that is, the functions containing a cell of level $\ell + 1$ that is nor active nor deactivated. We finally add in line 15 the remaining candidate functions to the list of active functions of level $\ell + 1$.

For the simplified hierarchical space we use Algorithm 4. The beginning of the algorithm is the same as

Algorithm 3 refine_hierarchical_space: standard hierarchical space

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}, \{\mathtt{NE}_\ell\}_{\ell=0}^{n}$
 1: if (number of levels of MESH) > (number of levels of SPACE) then
 2:     SPACE ← add_empty_level
 3: end if
 4: for $\ell = 0, \dots, n-1$ do
 5:     $\mathtt{F}^{\mathtt{A}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell} \setminus \mathtt{MF}_\ell$
 6:     $\mathtt{F}^{\mathtt{D}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{D}}_{\ell} \cup \mathtt{MF}_\ell$
 7:     $\mathtt{F}^{\mathtt{C}} \leftarrow$ get_basis_functions $(\mathtt{NE}_{\ell+1})$
 8:     $\mathtt{F}^{\mathtt{C}} \leftarrow \mathtt{F}^{\mathtt{C}} \setminus \mathtt{F}^{\mathtt{A}}_{\ell+1}$
 9:     for all $\beta \in \mathtt{F}^{\mathtt{C}}$ do
10:         $\mathtt{Q}_\beta \leftarrow$ get_cells $(\beta)$
11:         if (any $(\mathtt{Q}_\beta \notin \mathtt{E}^{\mathtt{A}}_{\ell+1} \cup \mathtt{E}^{\mathtt{D}}_{\ell+1})$) then
12:             $\mathtt{F}^{\mathtt{C}} \leftarrow \mathtt{F}^{\mathtt{C}} \setminus \beta$
13:         end if
14:     end for
15:     $\mathtt{F}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell+1} \cup \mathtt{F}^{\mathtt{C}}$
16: end for
    Output: SPACE                                        ▷ The refined space
```

for the standard space: we add an empty level if necessary, and for each level we remove the functions that have to be deactivated from the list of active and add them to the list of deactivated. The functions of the next level to be activated are the children of the marked functions that are not already active or deactivated. Lines 10–14 take into account a particular situation that may occur for the simplified space: it may happen that the support of a function that had never been activated has been already refined, as exemplified in Figure 8. In this case, this function is automatically marked to be deactivated.

Algorithm 4 refine_hierarchical_space: simplified hierarchical space

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}$
 1: if (number of levels of MESH) > (number of levels of SPACE) then
 2:     SPACE ← add_empty_level
 3: end if
 4: for $\ell = 0, \dots, n-1$ do
 5:     $\mathtt{F}^{\mathtt{A}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell} \setminus \mathtt{MF}_\ell$
 6:     $\mathtt{F}^{\mathtt{D}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{D}}_{\ell} \cup \mathtt{MF}_\ell$
 7:     $\mathtt{F}^{\mathtt{C}} \leftarrow$ get_children_of_function $(\mathtt{MF}_\ell)$
 8:     $\mathtt{F}^{\mathtt{C}} \leftarrow \mathtt{F}^{\mathtt{C}} \setminus (\mathtt{F}^{\mathtt{A}}_{\ell+1} \cup \mathtt{F}^{\mathtt{D}}_{\ell+1})$
 9:     $\mathtt{F}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell+1} \cup \mathtt{F}^{\mathtt{C}}$
10:     for all $\beta \in \mathtt{F}^{\mathtt{C}}$ do          ▷ Mark new functions whose support is already refined
11:         if (get_cells $(\beta) \cap \mathtt{E}^{\mathtt{A}}_{\ell+1} = \emptyset$) then
12:             $\mathtt{MF}_{\ell+1} \leftarrow \mathtt{MF}_{\ell+1} \cup \beta$
13:         end if
14:     end for
15: end for
    Output: SPACE                                        ▷ The refined space
```

Remark 7. *The refinement for the standard hierarchical space in Algorithm 3 can be improved by adding, between lines 6 and 7, the lines 7–9 of Algorithm 4 to activate the children of marked functions. This reduces the list of candidate functions for which one should check the support, and it is the way we implemented it*

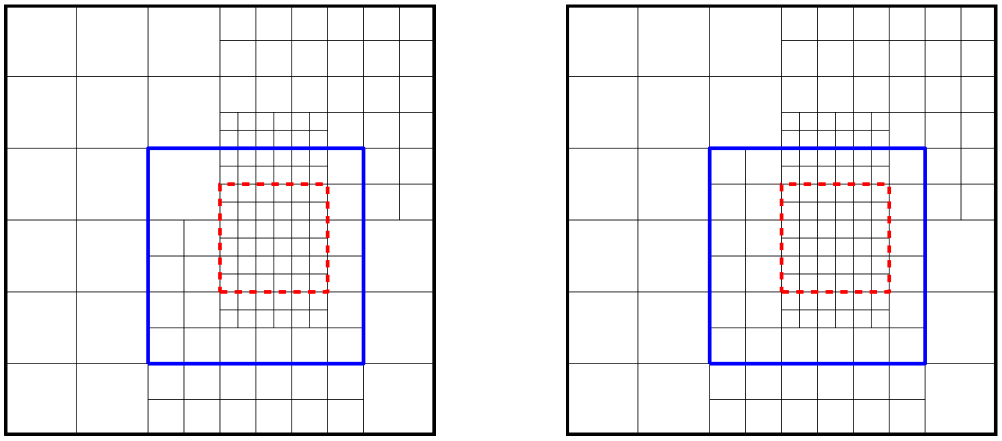


Figure 8: Example of a particular situation for the simplified hierarchical basis with biquadratic B-splines of maximum smoothness. In the mesh $\mathcal{Q}$ on the left the highlighted function of level 0 (solid blue line) is active, and the highlighted function of level 1 (dashed red line) has never been activated, because it is not a child of any function completely supported in Ω_1. After marking the highlighted function of level 0 (solid blue line) to obtain the mesh $\mathcal{Q}^*$ on the right, the level 1 function has to be immediately deactivated, because its support is contained in Ω_2^*.

in §7.1 (see also [12, Algorithm 2]). Notice that the check of lines 10–14 in Algorithm 4 is not necessary in this case.

It is also worth noting that the refinement described in Algorithm 1 can be used to construct the hierarchical basis $\mathcal{H}$ for a given hierarchical mesh, provided that we are given the list of active cells for each level. The procedure is described in Algorithm 5. One starts from a Cartesian grid and the basis of a tensor product space, and then proceeds level by level marking the active cells of the finest level that are not in the list of active cells given as an input, and therefore have to be refined. Obviously, the algorithm can be improved replacing the call to **refine** by a specialized function, that takes into account that at each step only elements of the finest level have been marked.

Algorithm 5 build_hierarchical_space

```
   Input: $\{\widehat{\mathtt{E}}^{\mathtt{A}}_\ell\}$
1: $\mathtt{MESH} \leftarrow$ mesh_cartesian          ▷ Initialize as a Cartesian grid of level 0
2: $\mathtt{SPACE} \leftarrow$ spline_space        ▷ Initialize as a tensor product space of level 0
3: for $k = 0, \dots, n-2$ do
4:     $\{\mathtt{MARKED}_\ell\}_{\ell=0}^{k-1} \leftarrow \emptyset$
5:     $\mathtt{MARKED}_k \leftarrow \mathtt{E}^{\mathtt{A}}_k \setminus \widehat{\mathtt{E}}^{\mathtt{A}}_k$
6:     $[\mathtt{MESH}, \mathtt{SPACE}] \leftarrow$ refine $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}\}_{\ell=0}^{k})$
7: end for
   Output: MESH, SPACE
```

6.2. Coarsening algorithms

We recall that for coarsening the set $\mathtt{MARKED}_\ell$ represents the list of entities of level ℓ that we want to reactivate, which are either deactivated cells whose children are all active (see (11)), or deactivated basis functions that have within their support at least one of such deactivated cells (see (14)). Using this set of candidates to reactivate, Algorithm 6 summarizes the main steps to provide the coarsened hierarchical mesh and space, that is, the structures for the mesh $\mathcal{Q}^-$ and the basis $\mathcal{H}^-$, respectively. The algorithm is divided in two main parts: in the first part we compute the cells that have to be reactivated, and coarsen the hierarchical mesh, updating the sets of active and deactivated cells of each level; in the second part, we compute the set of functions that have to be reactivated, and coarsen the hierarchical space updating the active and deactivated basis functions. In the following paragraphs we explain each of these steps in detail.

Algorithm 6 coarsen: update `MESH` and `SPACE` when shrinking the current subdomains using the candidates to reactivate (either cells or functions) given in $\{\mathtt{MARKED}_\ell\}$

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
 1: if (Marking cells) then
 2:     $\{\mathtt{ME}_\ell\} \leftarrow \{\mathtt{MARKED}_\ell\}$
 3: else if (Marking functions) then
 4:     $\{\mathtt{ME}_\ell\} \leftarrow$ compute_cells_to_reactivate $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\})$
 5: end if
 6: $[\mathtt{MESH}, \{\mathtt{EtD}_\ell\}] \leftarrow$ coarsen_hierarchical_mesh $(\mathtt{MESH}, \{\mathtt{ME}_\ell\})$
 7: $\{\mathtt{MF}_\ell\} \leftarrow$ functions_to_reactivate_from_cells $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{ME}_\ell\})$
 8: $\mathtt{SPACE} \leftarrow$ coarsen_hierarchical_space $(\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}, \{\mathtt{EtD}_\ell\})$
    Output: $\mathtt{MESH}, \mathtt{SPACE}$
```

Coarsening of the hierarchical mesh. First of all, we have to compute the cells that will be indeed reactivated, that we denote by $\{\mathtt{ME}_\ell\}$. In the case of marking cells this set is equal to $\{\mathtt{MARKED}_\ell\}$, since all the marked cells are admissible to be reactivated. In the case of marking functions we compute the cells to be reactivated according to the selection criteria in (15), the details are given in Algorithm 13 of Appendix A.

Once we have the set of cells to be reactivated, we can perform the coarsening of the mesh as described in Algorithm 7. We start by computing the children of the cells that will be reactivated, that are cells to be deleted from the list of active cells, and collect them in $\{\mathtt{EtD}_\ell\}$. We then remove the cells to be reactivated from the list of deactivated cells and add them to the list of active cells. Finally, if the finest level remains empty, that is, it does not contain any active element, it is removed.

Algorithm 7 coarsen_hierarchical_mesh

```
    Input: $\mathtt{MESH}, \{\mathtt{ME}_\ell\}$
 1: for $\ell = n-2, \dots, 0$ do
 2:     $\mathtt{EtD}_{\ell+1} \leftarrow$ get_children_of_cell$(\mathtt{ME}_\ell)$
 3:     $\mathtt{E}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{E}^{\mathtt{A}}_{\ell+1} \setminus \mathtt{EtD}_{\ell+1}$
 4:     $\mathtt{E}^{\mathtt{D}}_{\ell} \leftarrow \mathtt{E}^{\mathtt{D}}_{\ell} \setminus \mathtt{ME}_\ell$
 5:     $\mathtt{E}^{\mathtt{A}}_{\ell} \leftarrow \mathtt{E}^{\mathtt{A}}_{\ell} \cup \mathtt{ME}_\ell$
 6: end for
 7: $\mathtt{EtD}_0 \leftarrow \emptyset$
 8: if $\mathtt{E}^{\mathtt{A}}_{n-1} = \emptyset$ then
 9:     $\mathtt{MESH} \leftarrow$ remove_empty_level
10: end if
    Output: $\mathtt{MESH}, \{\mathtt{EtD}_\ell\}$
```

Coarsening of the space. Once the mesh information has been updated, we proceed with the coarsening of the discrete space. We start with the computation, in Algorithm 14 of Appendix A, of the functions that indeed will be reactivated. Notice that the computation in Algorithm 14 is also needed when marking basis functions, because as we have seen in §4.1.2 it may happen that some marked functions are not reactivated.

After selecting the functions that have to be reactivated, we coarsen the hierarchical space, that is, we update the sets of active and deactivated basis functions for each level. This is done in Algorithm 8 for the standard hierarchical space, and in Algorithm 9 for the simplified hierarchical space. The difference between both is how to decide the active functions to be removed: for the standard space active functions with one removed element of the same level in their supports have to be removed, as is done in line 2 of

Algorithm 8; for the simplified space the functions to be removed are the children of reactivated functions such that they are not children of any function that remains deactivated. This check is performed in lines 4-7 of Algorithm 9.

Algorithm 8 coarsen_hierarchical_space: standard hierarchical space

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}, \{\mathtt{EtD}_\ell\}$
 1: for $\ell = n-2, \dots, 0$ do
 2:     $\mathtt{F}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell+1} \setminus$ get_basis_functions $(\mathtt{EtD}_{\ell+1})$
 3:     $\mathtt{F}^{\mathtt{A}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell} \cup \mathtt{MF}_\ell$
 4:     $\mathtt{F}^{\mathtt{D}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{D}}_{\ell} \setminus \mathtt{MF}_\ell$
 5: end for
 6: if (number of levels of MESH) $<$ (number of levels of SPACE) then
 7:     SPACE $\leftarrow$ remove_empty_level
 8: end if
    Output: SPACE                                        ▷ The coarsened space
```

Algorithm 9 coarsen_hierarchical_space: simplified hierarchical space

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MF}_\ell\}$
 1: for $\ell = n-2, \dots, 0$ do
 2:     $\mathtt{F}^{\mathtt{A}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell} \cup \mathtt{MF}_\ell$
 3:     $\mathtt{F}^{\mathtt{D}}_{\ell} \leftarrow \mathtt{F}^{\mathtt{D}}_{\ell} \setminus \mathtt{MF}_\ell$
 4:     $\mathtt{F}^{\mathtt{C}} \leftarrow$ get_children_of_function$(\mathtt{MF}_\ell)$
 5:     $\mathtt{F}^{\mathtt{N}} \leftarrow$ get_neighbors$(\mathtt{MF}_\ell)$
 6:     $\mathtt{F}^{\mathtt{C}} \leftarrow \mathtt{F}^{\mathtt{C}} \setminus$ get_children_of_function$(\mathtt{F}^{\mathtt{D}}_{\ell} \cap \mathtt{F}^{\mathtt{N}})$
 7:     $\mathtt{F}^{\mathtt{A}}_{\ell+1} \leftarrow \mathtt{F}^{\mathtt{A}}_{\ell+1} \setminus \mathtt{F}^{\mathtt{C}}$
 8: end for
 9: if (number of levels of MESH) $<$ (number of levels of SPACE) then
10:     SPACE $\leftarrow$ remove_empty_level
11: end if
    Output: SPACE                                        ▷ The coarsened space
```

7. Implementation and numerical tests

We now present a brief description of the implementation of the adaptive isogeometric methods based on hierarchical splines in an Octave/Matlab code, along with some numerical results to show the effectiveness of the method and the efficiency of our current implementation.

7.1. Implementation of hierarchical splines in GeoPDEs

All the algorithms introduced above, and some other functions required by adaptive isogeometric methods, have been implemented in the free Octave/Matlab package GeoPDEs [16, 17], and will be soon released under a GNU/GPL license in the GeoPDEs webpage[2]. The GeoPDEs software is based on two main classes: one for the mesh (msh_cartesian or msh_multipatch), and one for the discrete space (sp_scalar or sp_multipatch). These classes contain, for the tensor product spaces of each level, the methods and the information explained in §5.1, see [17] for the details.

[2]https://rafavzqz.github.io/geopdes/

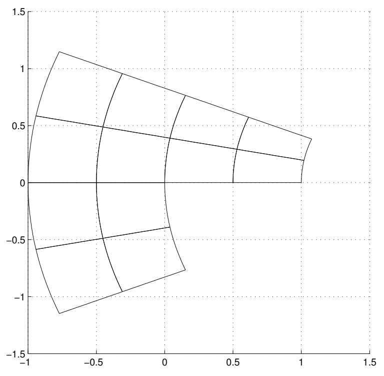


Figure 9: Curved L-shaped domain Ω and the initial mesh for the example of §7.2.1

For the implementation of hierarchical splines in GeoPDEs we have created two new classes: hierarchical_mesh and hierarchical_space, both of them with a multipatch counterpart (named with the _mp extension). These two classes contain all the information already introduced in §5, plus some other functionality useful for different purposes. For the convenience of potential users, we include in Appendix B four tables listing the properties and methods of the two classes.

All the algorithms presented in this paper have been implemented in the methods of the aforementioned classes. For instance, the refinement and coarsening algorithms from §6 are performed with the methods `hmsh_refine` and `hspace_refine`, and `hmsh_coarsen` and `hspace_coarsen`; and the assembly of the matrix presented in §3.2 is done with the method `op_u_v_hier` (and analogous), while the method `hspace_subdivision_matrix` computes the matrices C_ℓ, that are stored in the property `Csub` for convenience. Moreover, the classes contain other methods for postprocessing, that allow to plot the mesh, export the discrete solution to a `vtk` file, or to compute the discrete error when the exact solution is known.

7.2. Numerical tests

The implementation of our algorithms has been tested with several numerical examples, that we now explain in detail. Although the algorithms and the implementation are dimension independent, we have preferred to focus on two-dimensional problems, because their visualization is more intuitive.

7.2.1. Refinement in a curved L-shaped domain

In order to illustrate the performance of our local refinement tools, we apply an adaptive isogeometric method based on hierarchical splines to solve the problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ \quad u = g & \text{on } \partial\Omega \end{cases} \tag{20}$$

where f and g are chosen such that the exact solution u is given in polar coordinates by $u(\rho,\varphi) = \rho^{2/3}\sin(2\varphi/3)$, and Ω is the curved L-shaped domain shown in Figure 9. For the solution of the problem we apply a standard adaptive loop of the form

$$\text{SOLVE} \longrightarrow \text{ESTIMATE} \longrightarrow \text{MARK} \longrightarrow \text{REFINE},$$

considering two different a posteriori error estimators. We now briefly explain each step of the adaptive loop.

In the module SOLVE we apply the setting explained in §3.1 to our particular problem. Let $\mathcal{Q}$ and $\mathcal{H}$ denote the current hierarchical mesh and basis, respectively, and $W = \operatorname{span}\{\beta \circ \mathbf{F}^{-1}, \beta \in \mathcal{H}\}$. We want to compute the solution $u_\mathcal{H} \in W$ of the discrete problem

$$\int_\Omega \nabla u_\mathcal{H} \cdot \nabla v = \int_\Omega f v, \quad \forall\, v \in V,$$

where $V := \{v \in W,\ v|_{\partial\Omega} = 0\}$, and $u_{\mathcal{H}} = u_0 + u_g$, with $u_0 \in V$, and the lifting $u_g \in W$ such that $u_g|_{\partial\Omega} = g$ is computed with an L^2 projection on the boundary.

In the module ESTIMATE we consider two different *a posteriori error indicators*, the first one is element-based [26], and gives an indicator for each active element $Q \in \mathcal{Q}$, which corresponds to the active element $\mathbf{F}(Q)$ in the physical domain, defined by

$$\mathcal{E}_Q := h_Q \left(\int_{\mathbf{F}(Q)} |f + \Delta u_{\mathcal{H}}|^2 \right)^{\frac{1}{2}},$$

where $h_Q := \operatorname{diam}(\mathbf{F}(Q))$, whereas the second is function-based [24], and gives an indicator for each active basis function $\beta \in \mathcal{H}$, and the corresponding active function in the physical domain $\beta \circ \mathbf{F}^{-1}$, defined by

$$\mathcal{E}_\beta := h_\beta \sqrt{a_\beta} \left(\int_{\mathbf{F}(\operatorname{supp}\beta)} |f + \Delta u_{\mathcal{H}}|^2 (\beta \circ \mathbf{F}^{-1}) \right)^{\frac{1}{2}},$$

where $h_\beta := \operatorname{diam}(\mathbf{F}(\operatorname{supp}\beta))$ and a_β is the coefficient of the partition of unity in (5).

In the module MARK we use these estimators to compute a set $\mathcal{M}$ (either $\mathcal{M} := \mathcal{M}^e \subset \mathcal{Q}$ or $\mathcal{M} := \mathcal{M}^f \subset \mathcal{H}$) using the *maximum strategy* with parameter $\theta = 0.5$, i.e., $\mathcal{M}$ consists of the entities, active elements Q or basis functions β, such that

$$\mathcal{E}_Q \geq \theta \max_{Q' \in \mathcal{Q}} \mathcal{E}_{Q'}, \qquad \text{or} \qquad \mathcal{E}_\beta \geq \theta \max_{\beta' \in \mathcal{H}} \mathcal{E}_{\beta'}.$$

Finally, in the module REFINE, we use the marked set $\mathcal{M}$ to enlarge the hierarchy of subdomains as explained in §4.1.1, in order to get a finer hierarchical mesh and its corresponding hierarchical basis.

In Figure 10 we compare the energy error decay in terms of degrees of freedom, considering splines of maximum smoothness and degrees 2, 3, 4 and 5. For each case, we compare the behavior of tensor product B-splines refined globally with the two local adaptive refinement strategies just described above using hierarchical splines. Since the solution has a singularity in the reentrant corner of the domain Ω, the global refinement does not provide optimal order of convergence, but both adaptive strategies do.

Finally, we show some meshes obtained with the local refinement guided by the function-based error indicator in Figure 11, for biquadratic, bicubic and biquartic splines. In all cases, the error obtained is $\|\nabla(u - u_{\mathcal{H}})\|_{L^2(\Omega)} \approx 3.10^{-3}$.

7.2.2. Coarsening in the unit square

As a simple example, to test the performance of the algorithms for coarsening, we consider the linear elliptic problem (20), where the domain $\Omega = [0,1]^2$ is the unit square, and the data f and g are chosen such that the exact solution u is given by $u(x,y) = \tan^{-1}(25(x-y))$, which is shown in Figure 12. For this example we start from a very fine mesh, and apply the following adaptive coarsening loop

$$\text{SOLVE} \longrightarrow \text{ESTIMATE} \longrightarrow \text{MARK} \longrightarrow \text{COARSEN},$$

where the modules SOLVE and ESTIMATE are as explained in §7.2.1, in the module MARK, that we detail below, we use the output of ESTIMATE to select a set of elements or functions for coarsening (or reactivation), and in the module COARSEN we use the marked set to shrink the hierarchy of subdomains as explained in §4.1.2, in order to get a coarser hierarchical mesh and its corresponding hierarchical basis.

In the module MARK a list of elements or functions to be reactivated must be selected from the results of the a posteriori error estimators, that work on active entities. We start collecting a fixed percentage of active entities for which the estimated error is small: we denote by $N_{\mathcal{Q}}$ and N the number of active elements and functions, respectively, and for a fixed parameter $\theta \in (0,1)$ we set $\mathcal{M}^A$ as the $\lceil \theta N_{\mathcal{Q}} \rceil$ active elements or the $\lceil \theta N \rceil$ active functions with the smallest estimators. Let $\mathcal{M}^A_\ell := \mathcal{M}^A \cap \mathcal{Q}_\ell$ for the element-based estimators, and $\mathcal{M}^A_\ell := \mathcal{M}^A \cap \mathcal{B}_\ell$ for the function-based estimators. We give now a possible way to mark the candidates to reactivate for each case.

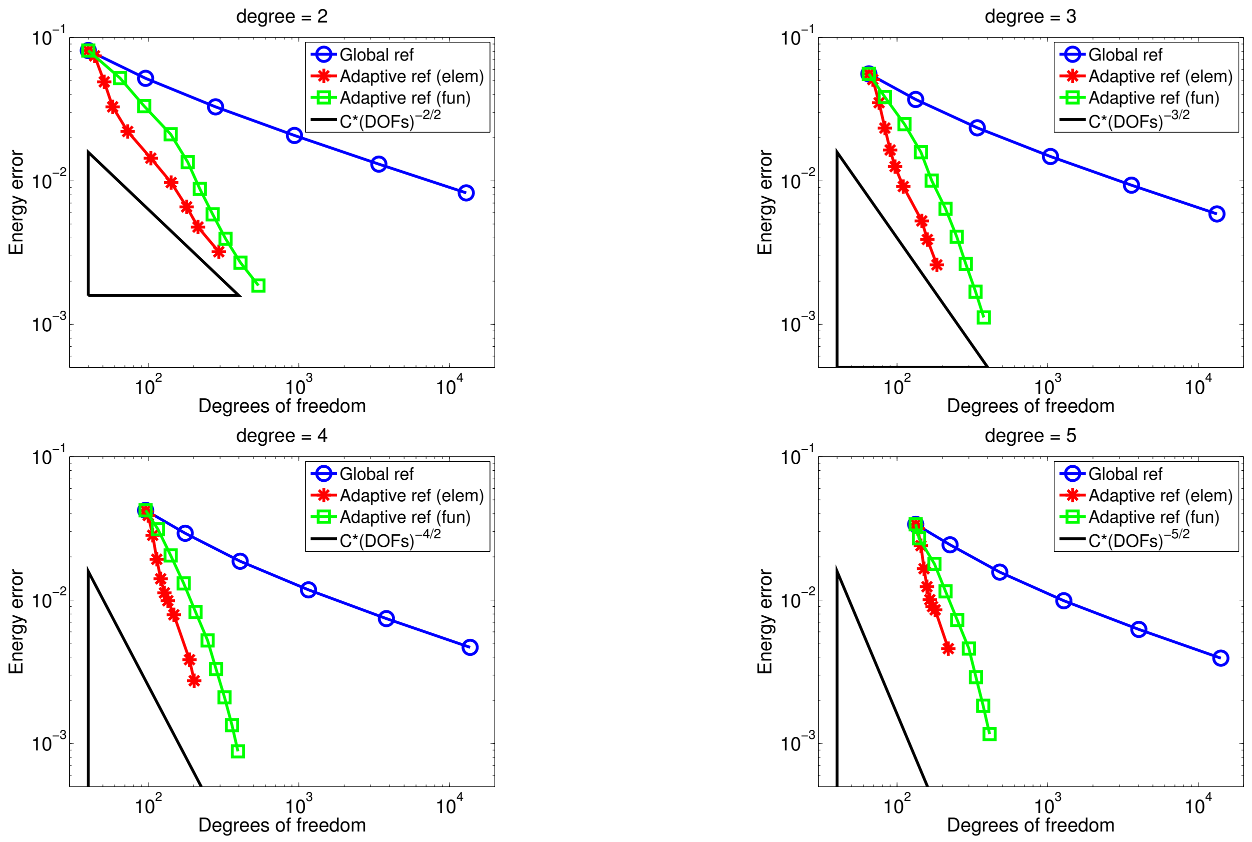


Figure 10: Energy error decay vs. degrees of freedom, using tensor product splines and hierarchical splines, i.e., global refinement and local adaptive refinement guided by two different a posteriori error indicators.

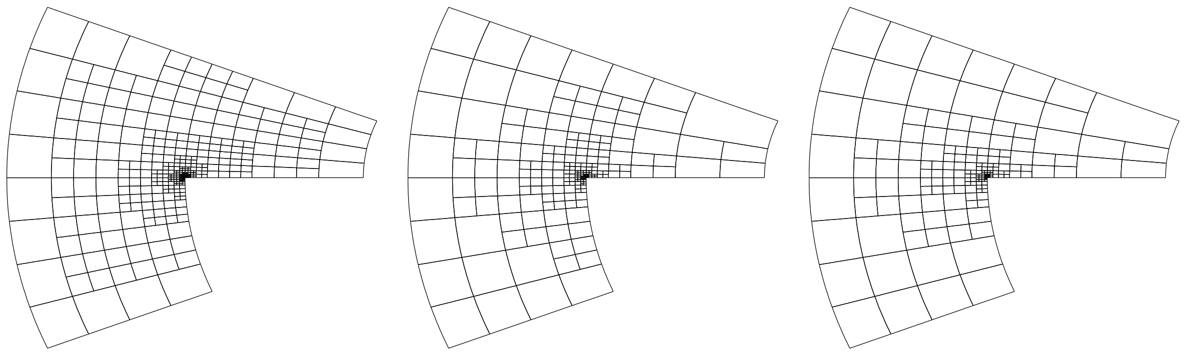
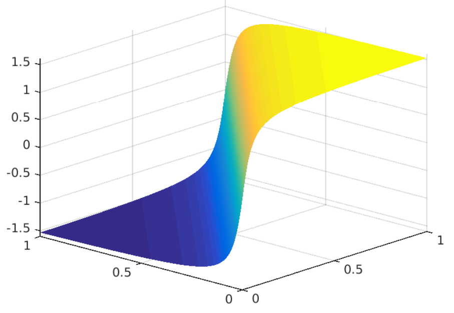


Figure 11: Comparison of meshes obtained with the function-based error indicator for different polynomial degrees; biquadratic with 9 levels, 396 elements and 410 DOFs (left), bicubic with 8 levels, 234 elements and 286 DOFs (middle) and biquartic with 7 levels, 183 elements and 282 DOFs (right).



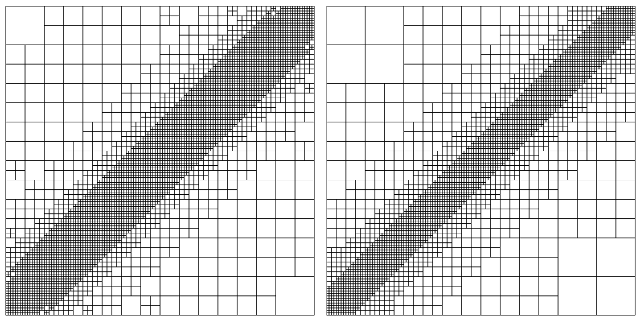


Figure 12: Exact solution of the problem considered for coarsening (left), and the final meshes obtained after coarsening marking by elements (middle) and by functions (right), with $\theta = 0.3$.

Table 1: Results of the coarsening algorithm marking deactivated elements (left) or deactivated functions (right) with $\theta = 0.3$. (C.Lev.: coarsest level).

DOFs	$N_{\mathcal{Q}}$	error	C.Lev.
17161	16384	0.00146624	8
13183	12814	0.00146624	7
10267	10162	0.00146624	6
8143	8218	0.00146628	5
6451	6754	0.00146678	5
4999	5608	0.00147311	4
4471	4858	0.00149726	4

DOFs	$N_{\mathcal{Q}}$	error	C.Lev.
17161	16384	0.00146624	8
13399	13018	0.00146624	7
10231	10150	0.00146624	6
7711	7810	0.00146632	5
5905	6112	0.00146757	5
4429	4732	0.00148676	4
3307	3670	0.00167448	4

- **Marking cells to reactivate:** If $\mathcal{M}^A_{\ell+1}$ is a set of active cells of level $\ell+1$, we define $\mathcal{M}^e_\ell \subset \mathcal{D}^e_\ell$ as the set of the deactivated cells of level ℓ such that *all their children* belong to $\mathcal{M}^A_{\ell+1}$, i.e.,

$$\mathcal{M}^e_\ell := \{Q \in \mathcal{Q}_\ell \mid \forall Q' \in \mathcal{Q}_{\ell+1}, (Q' \subset Q \Rightarrow Q' \in \mathcal{M}^A_{\ell+1})\}.$$

- **Marking functions to reactivate:** If $\mathcal{M}^A_{\ell+1}$ is a set of active basis functions of level $\ell+1$, we define $\mathcal{M}^f_\ell \subset \mathcal{D}^f_\ell$ as the set of functions admissible for reactivation of level ℓ such that *at least one child* belongs to $\mathcal{M}^A_{\ell+1}$, i.e.,

$$\mathcal{M}^f_\ell := \{\beta \in \mathcal{D}^f_\ell \mid \exists \beta' \in \mathcal{C}(\beta) \cap \mathcal{M}^A_{\ell+1}\}.$$

For the numerical tests, we consider bicubic splines with maximum smoothness, and start from a uniform Cartesian mesh of 16384 elements, corresponding to a 8-level hierarchical mesh with $2^7 = 128$ subdivisions in each direction. We apply the two strategies of coarsening, marking by elements and marking by functions, both of them with the values $\theta = 0.3$ and $\theta = 0.5$. In Tables 1 and 2 we show, after each coarsening step, the number of active functions and elements, the error in H^1 seminorm, and the coarsest level for which there exist active functions. From the results in the tables, we see that all the strategies reduce the amount of degrees of freedom to around one fourth without significantly affecting the accuracy. Further coarsening increases the error, specially for higher values of θ. We remark that, in all the tests, the finest level function at every step is of level 8.

In Figure 12 we show the final meshes obtained when marking by elements (middle) and by functions (right), with $\theta = 0.3$. It can be seen that marking by elements may lead to isolated elements, that do not activate any function of their level, while marking by functions produces a "cleaner" mesh. Finally, to see how the algorithm works, we show in Figure 13 the meshes corresponding to the last three steps of the coarsening process when marking by functions with $\theta = 0.5$.

Table 2: Results of the coarsening algorithm marking deactivated elements (left) or deactivated functions (right) with $\theta = 0.5$. (C.Lev.: coarsest level).

DOFs	N_Q	error	C.Lev.
17161	16384	0.00146624	8
10693	10444	0.00146624	7
6631	6730	0.00146676	6
4249	4552	0.00150103	5
3043	3466	0.00185167	4

DOFs	N_Q	error	C.Lev.
17161	16384	0.00146624	8
10969	10708	0.00146624	7
6955	7042	0.00146655	6
4285	4588	0.00149792	5
2347	2746	0.00236409	4

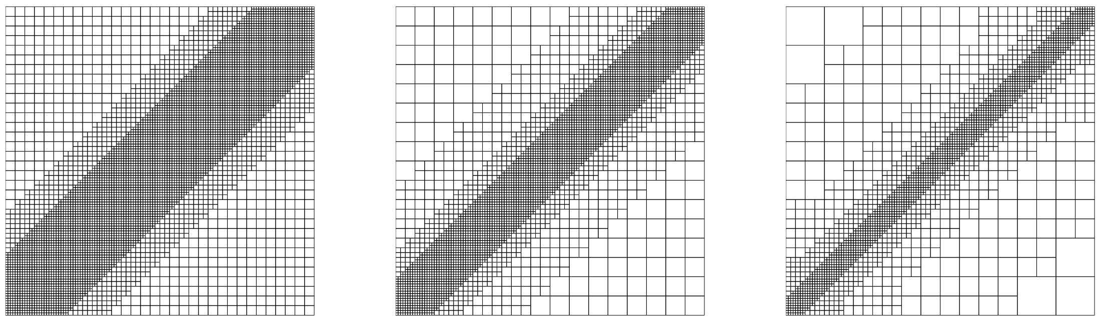


Figure 13: Meshes obtained at the last three steps of coarsening marking by functions, with $\theta = 0.5$.

7.2.3. Computational complexity of matrix and vector assembly

To evaluate the efficiency of our implementation we have measured the computational time, for the numerical test in the curved L-shaped domain described above, of the stiffness matrix and the load vector assembly computed as in §3.2. We have used as a reference the computational time obtained for the assembly in the standard version of GeoPDEs, using tensor product spaces with uniform refinement. This computational time is compared with the one obtained for the implementation of hierarchical B-splines in two cases: using an adaptive algorithm, like in §7.2.1, or refining uniformly, as for tensor product B-splines.

The results in Figure 14 present the computational time versus the number of levels, for degrees from 2 to 5. We remark that, in all cases, the magnitude of the energy error obtained with the hierarchical space of k levels is similar to the one obtained in the full tensor product space of the same level. As expected, the standard tensor product implementation beats the hierarchical implementation with uniform refinement, because it exploits the tensor product structure of the basis functions to reduce the computational effort. It is worth to note that both implementations behave asymptotically in the same way. The computational cost is clearly reduced when using the adaptive method, since the space has many less degrees of freedom.

7.2.4. Memory consumption

In the last set of tests we study the performance of our implementation in terms of memory consumption, which is computed as the maximum amount of memory used by Octave, and obtained with the **getrusage** built-in function. In all the computations we measure the necessary memory to generate the hierarchical mesh and the hierarchical space objects of GeoPDEs. This includes the computation of the matrices C_ℓ of §3.2, and of the refinement matrix K of §4.3 each time we add a new level. As already mentioned in Remark 3 some of these calculations could be performed on the fly, reducing the memory requirements, but they would increase the computational time in our Octave/Matlab implementation. Finally, we remark that for coarse meshes the amount of memory used is dominated by the required memory to simply run Octave[3].

[3]This could be avoided computing the memory used by the stored variables, instead of using **getrusage**, but it would not take into account the auxiliary matrices needed during the computation of C_ℓ and K.

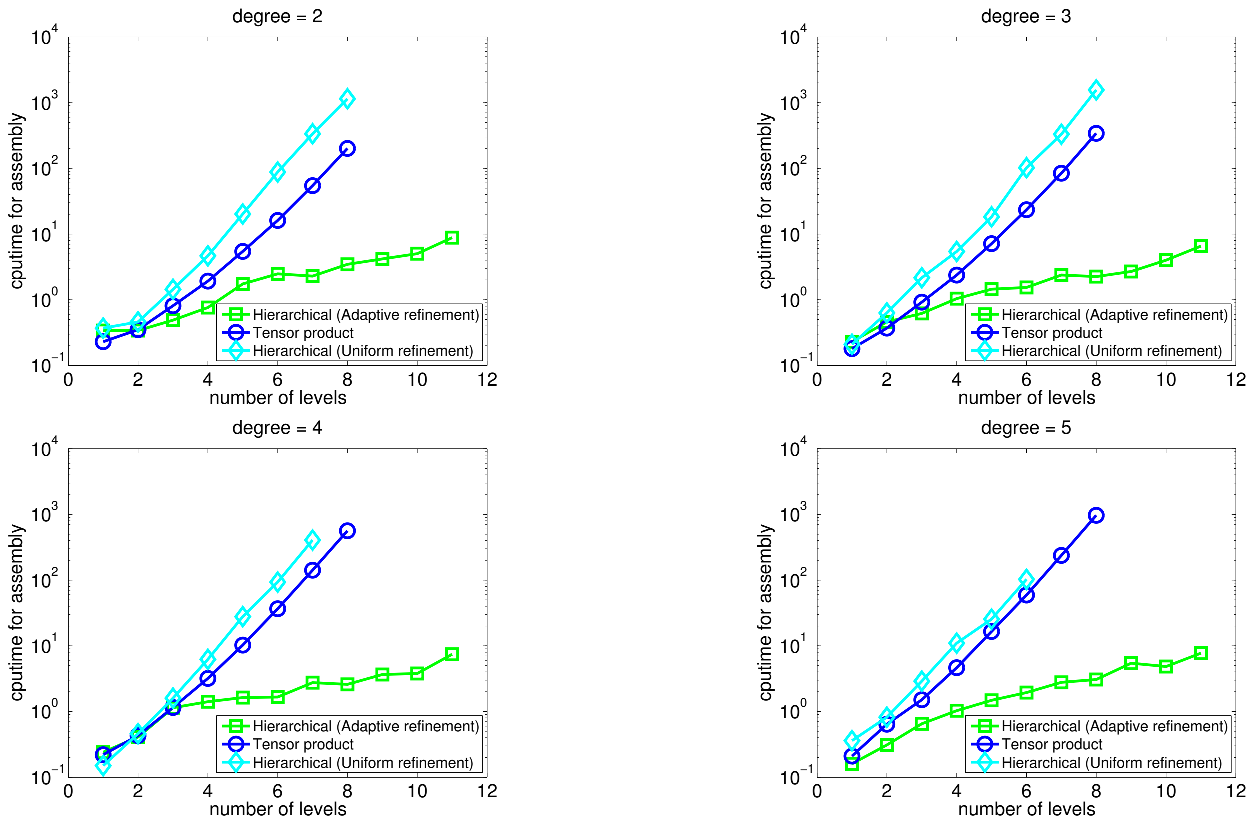


Figure 14: Computational time for the assembly versus the number of levels, using tensor product splines and hierarchical splines with uniform and adaptive refinement.

We start comparing the memory consumption for hierarchical splines and THB-splines, running in our implementation the same tests presented in [14] for the C++ G+SMO library. We consider the unit square domain refined near the diagonal, and generate four different sequences of meshes that differ on the mesh grading, as shown in Figure 15. For each grading we refine up to the tenth level, and compute hierarchical splines of degrees from 2 to 5, with maximum continuity.

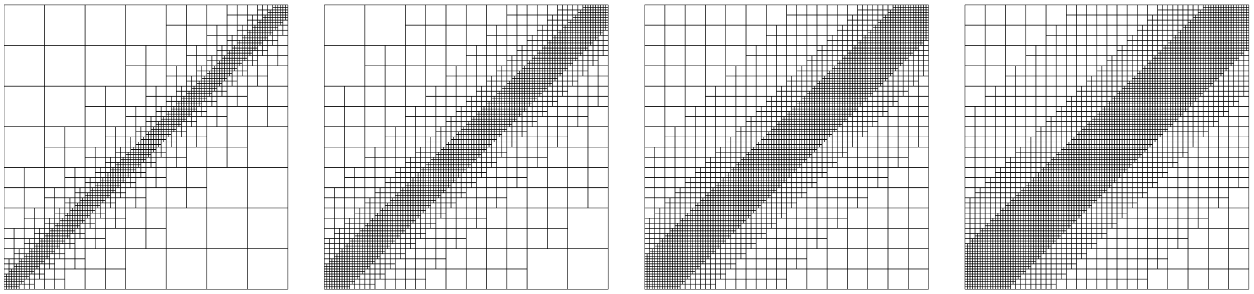


Figure 15: Four different hierarchical mesh configurations for the diagonal refinement of the unit square with increasing distance between cells of different levels (a–d). These configurations are the same from [14, Fig. 6].

The results are shown in Figure 16. We observe that the memory usage grows linearly both for hierarchical splines and THB-splines, independently of the degree and the mesh grading. This is in contrast with the results for G+SMO in [14], where the linear growth for THB-splines is only attained for highly graded meshes. This difference is due to the way we compute and store the coefficients for truncation. As explained in §2.4, a function of level ℓ truncated by a finer function of level $k \geq \ell$ is represented as a linear combination of functions of level k, but its restriction to an element of level k', with $\ell \leq k' < k$, can be expressed using only functions of level k'. In our implementation we take advantage of this fact and, as mentioned in Remark 2, we only compute the rows of the matrix $C_{k'}$ related to functions on active and deactivated elements of level k'. Instead, in [14] the function is always evaluated from the coefficients of functions of level k, which increases the memory consumption. Notice also that this behavior gets worse when increasing the degree or the dimension. Moreover, in our case THB-splines consume less memory than hierarchical splines, due to the lower number of nonzero entries in these matrices.

It is also worth noting that the consumed memory is higher, in terms of degrees of freedom, for the less graded meshes, which suggests that the amount of memory used depends more on the number of levels than on the number of degrees of freedom. To verify this, we have run a similar set of tests with a different refinement pattern, refining towards a corner as in the meshes of Figure 17. The obtained results, that we show in Figure 18 (left) for the bicubic case and for standard hierarchical B-splines (not truncated), show that the memory increase is not linear in terms of the degrees of freedom anymore. As can be seen in Figure 18 (right), for each refinement pattern the amount of memory used is mainly dependent on the number of levels, while the curves remarkably differ between the two refinement patterns. The worst results, in terms of memory consumption, are obtained for the uniform refinement case.

8. Conclusions

We have introduced the data structures and algorithms to perform refinement and coarsening of hierarchical splines. Our algorithms cover the cases of standard or simplified hierarchical splines, with or without truncation. In particular, we have presented a new method to apply coarsening that, unlike previously existing methods, can be understood as the inverse of refinement. All the algorithms have been implemented in the open-source Octave code GeoPDEs, and tested in simple numerical examples. We plan to analyse different coarsening strategies in a future work.

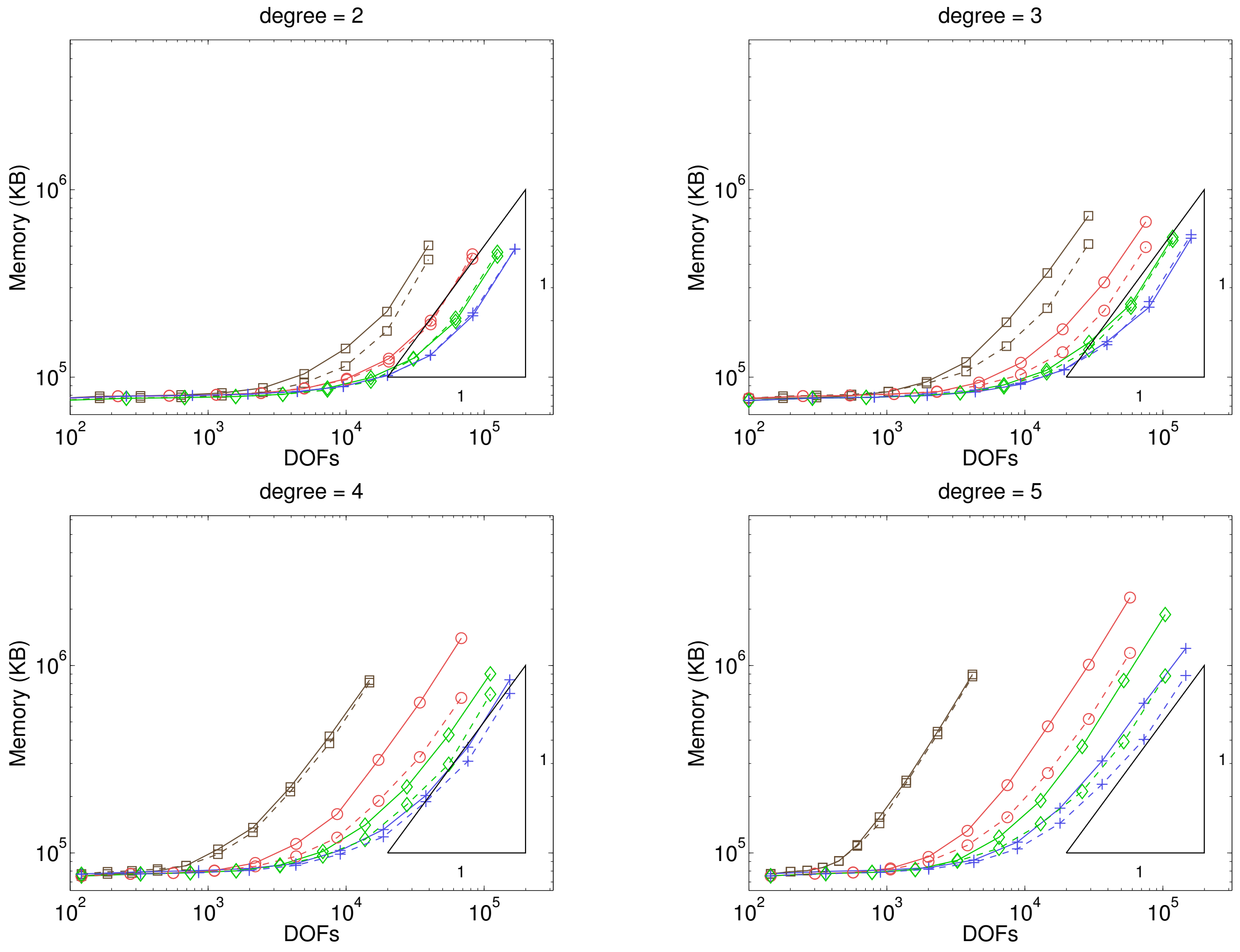


Figure 16: Experimental results for hierarchical splines (solid lines) and THB-splines (dashed lines) for the four hierarchical mesh configurations shown in Figure 15. The brown, red, green and blue colors (square, circle, diamond and plus markers) correspond to the meshes (a–d), respectively.

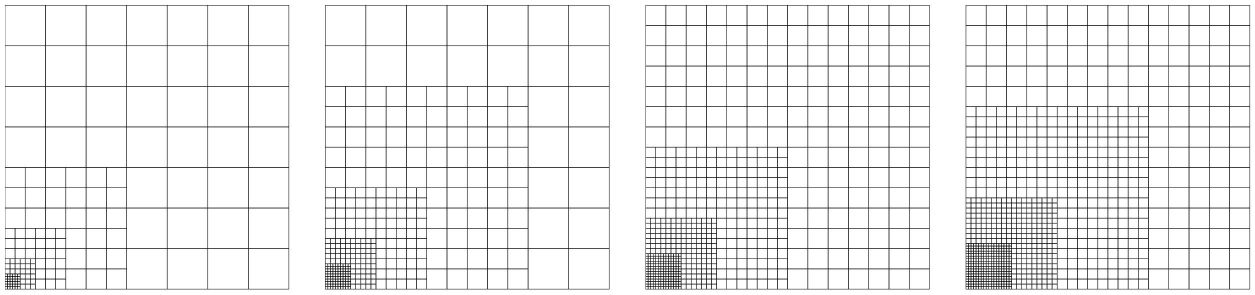
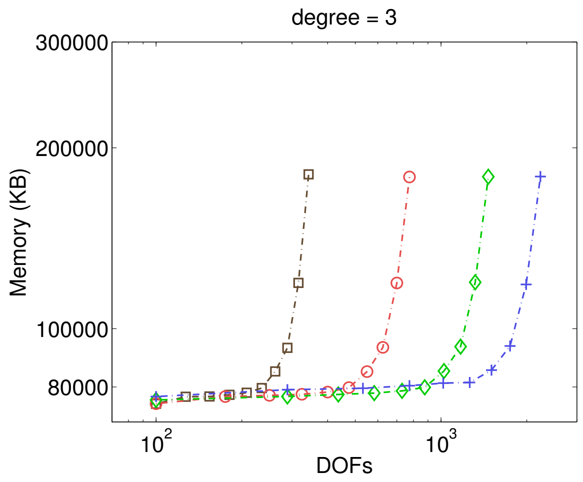


Figure 17: Four different hierarchical mesh configurations for a corner refinement of the unit square with increasing distance between cells of different levels (a–d).



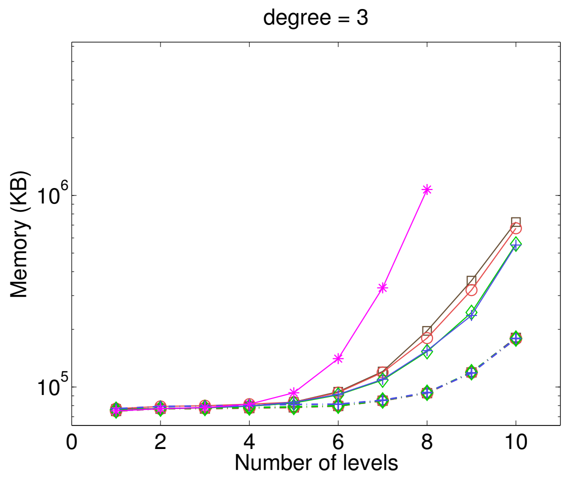


Figure 18: On the left, we show the memory consumption in terms of DOFs for the four hierarchical mesh configurations for the corner refinement given in Figure 17. The brown, red, green and blue colors (square, circle, diamond and plus markers) correspond to the meshes (a–d), respectively. On the right, we compare this memory consumption (dashed-dotted line) in terms of the amount of levels with the corresponding one (solid line) to the hierarchical mesh configurations for the diagonal refinement given in Figure 15. The solid purple line with the star marker corresponds to the uniform refinement case.

Acknowledgements

The authors would like to thank Cesare Bracco for his help in the implementation of truncated functions, specially regarding the matrix of Section 4.3, and for beta testing. This work was partially supported by the Argentina-Italy bilateral project "Simulation of biomolecules and biomembranes: a challenge for numerical analysis" funded by CNR and CONICET (2015-2016). Eduardo M. Garau was partially supported by CONICET through grant PIP 112-2011-0100742, by Universidad Nacional del Litoral through grants CAI+D 500 201101 00029 LI, 501 201101 00476 LI, by Agencia Nacional de Promoción Científica y Tecnológica, through grants PICT-2012-2590 and PICT-2014-2522 (Argentina). Rafael Vázquez was partially supported by the European Research Council through the FP7 ERC Consolidator Grant no. 616563 HIGEOM, by European Union's Horizon 2020 research and innovation programme through the grant no. 680448 CAxMan, and by the Italian Ministry of Education, University and Research through the project PRIN-2012HBLYE4. This support is gratefully acknowledged.

Appendix A. Detailed algorithms for refinement and coarsening

For completeness, we detail here the missing algorithms needed for refinement and coarsening.

Algorithms used during refinement

Algorithm 10 computes the set of cells to be deactivated during refinement. In the case of marking cells nothing has to be done, since the marked cells and the ones to be deactivated coincide. In the case of marking basis functions, for each level we consider the marked basis functions, and collect the active cells of that level within their supports.

Algorithm 10 compute_cells_to_refine

```
   **Input:** $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
1: **for** $\ell = 0, \dots, n-1$ **do**
2:     $\mathtt{ME}_\ell \leftarrow$ **get_cells** $(\mathtt{MARKED}_\ell)$
3:     $\mathtt{ME}_\ell \leftarrow \mathtt{ME}_\ell \cap \mathtt{E}_\ell^\mathtt{A}$
4: **end for**
   **Output:** $\{\mathtt{ME}_\ell\}$
```

Analogously, it is necessary to collect the basis functions that have to be deactivated during refinement, that is, those functions whose support has no active cell of their same level in the refined mesh. In the case of marking cells, this is done in Algorithm 11: for each level, we first collect the active functions that do not vanish in the marked cells, and then, for each of these functions, we keep as non-marked functions those that have at least one active cell within their support.

Algorithm 11 functions_to_deactivate_from_cells

```
   **Input:** $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
1: **for** $\ell = 0, \dots, n-1$ **do**
2:     $\mathtt{MF}_\ell \leftarrow$ **get_basis_functions** $(\mathtt{MARKED}_\ell)$
3:     $\mathtt{MF}_\ell \leftarrow \mathtt{MF}_\ell \cap \mathtt{F}_\ell^\mathtt{A}$
4:     **for all** $\beta \in \mathtt{MF}_\ell$ **do**
5:         **if** (**get_cells** $(\beta) \cap \mathtt{E}_\ell^\mathtt{A} \neq \emptyset$) **then**
6:             $\mathtt{MF}_\ell \leftarrow \mathtt{MF}_\ell \setminus \beta$
7:         **end if**
8:     **end for**
9: **end for**
   **Output:** $\{\mathtt{MF}_\ell\}$
```

In the case of marking basis functions, it may happen that more functions than those marked have to be deactivated, as in the example of Figure 4. This set is computed in Algorithm 12: for each level we first collect, using the function **get_neighbors**, the set of active functions such that their support shares at least one cell with the one of a marked function. Then, as it is done when marking cells, we keep as non-marked the functions that have at least one active cell within their support. This check can be done only for functions that were not already given as marked in the input, to save computational time.

Algorithms used during coarsening

In the case of marking functions for coarsening, we compute in Algorithm 13 the cells to be reactivated, according to the selection criteria in (15): a cell Q of level ℓ to be reactivated must have all its children active, it must be in the support of a marked function, and it cannot be in the support of a deactivated function of the same level that has not been marked.

Algorithm 14 computes, from the list of cells to be reactivated, the list of functions that indeed will be reactivated. For the standard hierarchical space this is given by the deactivated functions that do not vanish in some reactivated cell. For the simplified hierarchical space we must also include, in the list of functions to be reactivated, deactivated functions such that after coarsening none of their parents will be deactivated, such as the one in Figure 8. As said above, the computation in Algorithm 14 is also needed

Algorithm 12 functions_to_deactivate_from_neighbors

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
 1: for $\ell = 0, \dots, n-1$ do
 2:     $\mathtt{MF}_\ell \leftarrow$ get_neighbors $(\mathtt{MARKED}_\ell)$
 3:     $\mathtt{MF}_\ell \leftarrow (\mathtt{MF}_\ell \cap \mathtt{F}_\ell^{\mathtt{A}}) \setminus \mathtt{MARKED}_\ell$        ▷ Check only the support of non-marked functions.
 4:     for all $\beta \in \mathtt{MF}_\ell$ do
 5:         if (get_cells $(\beta) \cap \mathtt{E}_\ell^{\mathtt{A}} \neq \emptyset$) then
 6:             $\mathtt{MF}_\ell \leftarrow \mathtt{MF}_\ell \setminus \beta$
 7:         end if
 8:     end for
 9:     $\mathtt{MF}_\ell \leftarrow \mathtt{MF}_\ell \cup \mathtt{MARKED}_\ell$
10: end for
    Output: $\{\mathtt{MF}_\ell\}$
```

Algorithm 13 compute_cells_to_reactivate

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{MARKED}_\ell\}$
 1: for $\ell = 0, \dots, n-2$ do
 2:     $\mathtt{ME}_\ell \leftarrow \emptyset$
 3:     $\mathtt{I}_\ell \leftarrow$ get_cells $(\mathtt{MARKED}_\ell)$
 4:     for $Q \in \mathtt{I}_\ell$ do
 5:         if (get_children_of_cell$(Q) \cap \mathtt{E}_{\ell+1}^{\mathtt{D}} = \emptyset$) then
 6:             if (get_basis_functions$(Q) \cap (\mathtt{F}_\ell^{\mathtt{D}} \setminus \mathtt{MARKED}_\ell) = \emptyset$) then
 7:                 $\mathtt{ME}_\ell \leftarrow \mathtt{ME}_\ell \cup Q$
 8:             end if
 9:         end if
10:     end for
11: end for
12: $\mathtt{ME}_{n-1} \leftarrow \emptyset$
    Output: $\{\mathtt{ME}_\ell\}$
```

when marking basis functions, because as we have seen in §4.1.2 it may happen that some marked functions are not reactivated. However, in this case the algorithm can be improved computing in line 3 the intersection only with marked functions, that should be given as an input.

Appendix B. Properties and methods of the classes in GeoPDEs

We list the properties and methods of the new hierarchical classes implemented in GeoPDEs. Tables B.3 and B.4 contain the properties of the classes that define the hierarchical mesh and the hierarchical space, respectively. Tables B.5 and B.6 contain the main methods for the same classes.

[1] T. J. R. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Engrg. 194 (39-41) (2005) 4135–4195, ISSN 0045-7825.

[2] J. A. Cottrell, T. J. R. Hughes, Y. Bazilevs, Isogeometric Analysis: toward integration of CAD and FEA, John Wiley & Sons, 2009.

[3] L. Beirão da Veiga, A. Buffa, G. Sangalli, R. Vázquez, Mathematical analysis of variational isogeometric methods, Acta Numer. 23 (2014) 157–287, ISSN 0962-4929, doi:10.1017/S096249291400004X.

[4] C. de Boor, A practical guide to splines, vol. 27 of *Applied Mathematical Sciences*, Springer-Verlag, New York, revised edn., ISBN 0-387-95366-3, 2001.

Algorithm 14 functions_to_reactivate_from_cells

```
    Input: $\mathtt{MESH}, \mathtt{SPACE}, \{\mathtt{ME}_\ell\}$
 1: $\{\mathtt{MF}_\ell\} \leftarrow \emptyset$
 2: for $\ell = 0, \dots, n-2$ do
 3:     $\mathtt{MF}_\ell \leftarrow \mathtt{MF}_\ell \cup$(get_basis_functions $(\mathtt{ME}_\ell) \cap \mathtt{F}^{\mathtt{D}}_\ell$)
 4:     if (Simplified hierarchical space) then
 5:         $\mathtt{C}^{\mathtt{D}} \leftarrow$ get_children_of_function($\mathtt{MF}_\ell$) $\cap\, \mathtt{F}^{\mathtt{D}}_{\ell+1}$
 6:         for $\beta \in \mathtt{C}^{\mathtt{D}}$ do
 7:             if (get_parents_of_function($\beta$) $\cap\, (\mathtt{F}^{\mathtt{D}}_\ell \setminus \mathtt{MF}_\ell) = \emptyset$) then
 8:                 $\mathtt{MF}_{\ell+1} \leftarrow \mathtt{MF}_{\ell+1} \cup \beta$
 9:             end if
10:         end for
11:     end if
12: end for
    Output: $\{\mathtt{MF}_\ell\}$
```

Table B.3: The properties of the hierarchical_mesh class.

Name	Type	Size	Description
`ndim`	Scalar	1×1	Dimension of the parametric domain
`rdim`	Scalar	1×1	Dimension of the physical space in which the domain is embedded
`nlevels`	Scalar	1×1	Number of levels of the hierarchical mesh
`nsub`	Array	1×ndim	Number of subintervals for *h*-refinement, in each parametric direction
`nel`	Scalar	1×1	Total number of elements of the mesh
`nel_per_level`	Array	1×nlevels	Number of active elements in each level
`active`	Cell array	1×nlevels	Indices of the active elements in each level
`deactivated`	Cell array	1×nlevels	Indices of deactivated elements in each level
`mesh_of_level`	msh_cartesian or msh_multipatch	1×nlevels	Cartesian grid (or multipatch grid) of each level
`msh_lev`	Cell array (struct)	1×nlevels	Quadrature and parametrization information for the active elements of each level
`boundary`	hierarchical_mesh	1×(2*ndim)	A mesh object for each boundary side

[5] L. L. Schumaker, Spline functions: basic theory, Cambridge Mathematical Library, Cambridge University Press, Cambridge, third edn., ISBN 978-0-521-70512-7, 2007.

[6] A.-V. Vuong, C. Giannelli, B. Jüttler, B. Simeon, A hierarchical approach to adaptive local refinement in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 200 (49-52) (2011) 3554–3567, ISSN 0045-7825, doi:10.1016/j.cma.2011.09.004.

[7] P. Hennig, S. Müller, M. Kästner, Bézier extraction and adaptive refinement of truncated hierarchical NURBS, Comput. Methods Appl. Mech. Engrg. 305 (2016) 316–339, ISSN 0045-7825, doi: 10.1016/j.cma.2016.03.009.

[8] G. Kuru, C. Verhoosel, K. van der Zee, E. van Brummelen, Goal-adaptive Isogeometric Analysis with hierarchical splines, Comput. Methods Appl. Mech. and Engrg. 270 (2014) 270–292, ISSN 0045-7825, doi:10.1016/j.cma.2013.11.026.

[9] D. Schillinger, L. Dedè, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, T. J. R. Hughes, An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS,

Table B.4: The properties of the hierarchical_space class.

Name	Type	Size	Description
`type`	String	1×1	Type of hierarchical space: either ‘standard’ or ‘simplified’
`truncated`	Logical	1×1	Decide whether to use the truncated basis or not
`nlevels`	Scalar	1×1	Number of levels of the hierarchical space (equal to the number of levels of the mesh)
`ndof`	Scalar	1×1	Total number of basis functions of the space
`ndof_per_level`	Array	1×nlevels	Number of active functions in each level
`active`	Cell array	1×nlevels	Indices of the active functions in each level
`deactivated`	Cell array	1×nlevels	Indices of the deactivated functions in each level
`space_of_level`	sp_scalar or sp_multipatch	1×nlevels	Tensor product space (or multipatch space) for each level
`Proj`	Cell array	(nlevels-1)×ndim (nlevels-1)×npatch	Subdivision matrices for the univariate spaces. For multipatch spaces, they are stored for each patch
`Csub`	Cell array	1×nlevels	Subdivision matrix C_ℓ, to write active functions as linear combinations of basis functions of finer levels
`coeff_pou`	Array	ndof×1	Coefficients for the partition of unity
`boundary`	hierarchical_space	1×(2*ndim)	A space object for each boundary side
`dofs`	Array	ndof×1	Only for boundary spaces, global numbering (in the bulk domain) of the basis functions on the boundary

Table B.5: The methods of the hierarchical_mesh class.

Method name	Output	Description
`hmsh_add_new_level`	hierarchical_mesh object	Add a new empty level, without active or deactivated elements
`hmsh_remove_empty_level`	hierarchical_mesh object	Remove the finest level if it is empty
`hmsh_refine`	hierarchical_mesh object	Refine the hierarchical mesh from a list of marked elements, as in Algorithm 2
`hmsh_coarsen`	hierarchical_mesh object	Coarsen the hierarchical mesh from a list of marked elements, as in Algorithm 7
`hmsh_get_children`	Array	Compute the children of a given element (or a list of elements)
`hmsh_get_parent`	Scalar	Compute the parent of a given element (or a list of elements)
`hmsh_plot_cells`	Figure	For curves and surfaces, draw the hierarchical mesh

immersed boundary methods, and T-spline CAD surfaces, Comput. Methods Appl. Mech. Engrg. 249-252 (2012) 116 – 150, ISSN 0045-7825, doi:10.1016/j.cma.2012.03.017.

[10] İ. Temizer, C. Hesch, Hierarchical NURBS in frictionless contact, Comput. Methods Appl. Mech. Engrg. 299 (2016) 161 – 186, ISSN 0045-7825, doi:10.1016/j.cma.2015.11.006.

[11] C. Apprich, K. Höllig, J. Hörner, A. Keller, E. Nava Yazdani, Finite Element Approximation with Hierarchical B-Splines, in: J.-D. Boissonnat, A. Cohen, O. Gibaru, C. Gout, T. Lyche, M.-L. Mazure,

Table B.6: The methods of the hierarchical_space class.

Method name	Output	Description
`hspace_add_new_level`	hierarchical_space object	Add a new empty level, without active or deactivated functions
`hspace_remove_empty_level`	hierarchical_space object	Remove the finest level, if it is empty
`hspace_refine`	hierarchical_space object	Refine the hierarchical space, as in Algorithms 3 and 4
`hspace_coarsen`	hierarchical_space object	Coarsen the hierarchical space, as in Algorithms 8 and 9
`hspace_get_children`	Array	Compute the children of a given basis function (or a list of functions)
`hspace_get_parents`	Array	Compute the parents of a given basis function (or a list of functions)
`hspace_subdivision_matrix`	Cell array	Compute the subdivision matrices C_ℓ as in Section 3.2
`hspace_eval_hmsh`	NDArray or cell array (see function help)	Evaluate the computed solution in the quadrature points of the hierarchical mesh
Other methods, analogous to the ones existing for tensor product spaces (see [17])		
`sp_drchlt_l2_proj, sp_eval, sp_h1_error, sp_l2_error, sp_to_vtk, op_u_v_hier, op_gradu_gradv_hier, op_f_v_hier`		

L. L. Schumaker (Eds.), Curves and Surfaces, vol. 9213 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, ISBN 978-3-319-22804-4, 1–15, doi:10.1007/978-3-319-22804-4_1, 2015.

[12] P. Bornemann, F. Cirak, A subdivision-based implementation of the hierarchical B-spline finite element method, Comput. Methods Appl. Mech. Engrg. 253 (2013) 584 – 598, ISSN 0045-7825, doi: 10.1016/j.cma.2012.06.023.

[13] M. Scott, D. Thomas, E. Evans, Isogeometric spline forests, Comput. Methods Appl. Mech. Engrg. 269 (0) (2014) 222 – 264, ISSN 0045-7825, doi:10.1016/j.cma.2013.10.024.

[14] C. Giannelli, B. Jüttler, S. K. Kleiss, A. Mantzaflaris, B. Simeon, J. Špeh, THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 299 (2016) 337 – 365, ISSN 0045-7825, doi:10.1016/j.cma.2015.11.002.

[15] G. Kiss, C. Giannelli, B. Jüttler, Algorithms and Data Structures for Truncated Hierarchical B-splines, in: M. Floater, T. Lyche, M.-L. Mazure, K. Mørken, L. L. Schumaker (Eds.), Mathematical Methods for Curves and Surfaces, vol. 8177 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, ISBN 978-3-642-54381-4, 304–323, doi:10.1007/978-3-642-54382-1_18, 2014.

[16] C. de Falco, A. Reali, R. Vázquez, GeoPDEs: a research tool for Isogeometric Analysis of PDEs, Adv. Engrg. Softw. 42 (12) (2011) 1020–1034, doi:10.1016/j.advengsoft.2011.06.010.

[17] R. Vázquez, A new design for the implementation of isogeometric analysis in Octave and Matlab: GeoPDEs 3.0, Comput. Math. Appl. (2016) –ISSN 0898-1221, doi:10.1016/j.camwa.2016.05.010, in press.

[18] A. Buffa, E. M. Garau, Refinable spaces and local approximation estimates for hierarchical splines, IMA J. Numer. Anal. doi:10.1093/imanum/drw035, in press.

[19] C. Giannelli, B. Jüttler, H. Speleers, THB-splines: The truncated basis for hierarchical splines, Comput. Aided Geom. Design. 29 (7) (2012) 485 – 498, ISSN 0167-8396, doi:10.1016/j.cagd.2012.03.025.

[20] C. Giannelli, B. Jüttler, H. Speleers, Strongly stable bases for adaptively refined multilevel spline spaces, Adv. Comput. Math. 40 (2) (2014) 459–490, ISSN 1019-7168, doi:10.1007/s10444-013-9315-2.

[21] P. Krysl, E. Grinspun, P. Schröder, Natural hierarchical refinement for finite element methods, Internat. J. Numer. Methods Engrg. 56 (8) (2003) 1109–1124, ISSN 0029-5981, doi:10.1002/nme.601.

[22] R. Kraft, Adaptive and linearly independent multilevel B-splines, in: Surface fitting and multiresolution methods (Chamonix–Mont-Blanc, 1996), Vanderbilt Univ. Press, Nashville, TN, 209–218, 1997.

[23] H. Speleers, C. Manni, Effortless quasi-interpolation in hierarchical spaces, Numer. Math. (2015) 1–30ISSN 0029-599X, doi:10.1007/s00211-015-0711-z.

[24] A. Buffa, E. M. Garau, A posteriori error estimators for hierarchical B-spline discretizations, arXiv:1611.07816 [math.NA], 2016.

[25] E. Grinspun, P. Krysl, P. Schröder, CHARMS: A Simple Framework for Adaptive Simulation, SIGGRAPH (ACM Transactions on Graphics) 21 (3) (2002) 281–290, doi:10.1145/566654.566578.

[26] A. Buffa, C. Giannelli, Adaptive isogeometric methods with hierarchical splines: Error estimator and convergence, Math. Models Methods Appl. Sci. 26 (01) (2016) 1–25, doi:10.1142/S0218202516500019.

[27] X. Wei, Y. Zhang, T. J. R. Hughes, M. A. Scott, Truncated hierarchical CatmullClark subdivision with local refinement, Comput. Methods Appl. Mech. Engrg. 291 (2015) 1–20, ISSN 0045-7825, doi: http://dx.doi.org/10.1016/j.cma.2015.03.019.

[28] A. Bressan, D. Mokriš, A versatile strategy for the implementation of adaptive splines, NFN technical report 50, Johannes Kepler Universität Linz, 2016.

[29] F. Cirak, Q. Long, Subdivision shells with exact boundary control and non-manifold geometry, Internat. J. Numer. Methods Engrg. 88 (9) (2011) 897–923, ISSN 1097-0207, doi:10.1002/nme.3206.

[30] F. Buchegger, B. Jüttler, A. Mantzaflaris, Adaptively refined multi-patch B-splines with enhanced smoothness, Appl. Math. Comput. 272, Part 1 (2016) 159 – 172, ISSN 0096-3003, doi: 10.1016/j.amc.2015.06.055, subdivision, Geometric and Algebraic Methods, Isogeometric Analysis and Refinability.

[31] W. Jiang, J. E. Dolbow, Adaptive refinement of hierarchical B-spline finite elements with an efficient data transfer algorithm, Internat. J. Numer. Methods Engrg. 102 (3-4) (2015) 233–256, ISSN 1097-0207, doi:10.1002/nme.4718.