Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 400 (2022) 115465
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Weighted quadrature for hierarchical B-splines

Carlotta Giannelli[a], Tadej Kanduč[b], Massimiliano Martinelli[c,*], Giancarlo Sangalli[d,c], Mattia Tani[c]

[a] *Dipartimento di Matematica e Informatica "U. Dini", Università degli Studi di Firenze, Italy*
[b] *Faculty of Mathematics and Physics and Faculty of Mechanical Engineering, University of Ljubljana, Ljubljana, Slovenia*
[c] *Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes" – CNR, Pavia, Italy*
[d] *Dipartimento di Matematica "F. Casorati", Università degli Studi di Pavia, Italy*

Received 19 October 2021; received in revised form 23 July 2022; accepted 23 July 2022
Available online xxxx

Abstract

We present weighted quadrature for hierarchical B-splines to address the fast formation of system matrices arising from adaptive isogeometric Galerkin methods with suitably graded hierarchical meshes. By exploiting a local tensor product structure, we extend the construction of weighted rules from the tensor product to the hierarchical spline setting. The proposed algorithm has a computational cost proportional to the number of degrees of freedom and advantageous properties with increasing spline degree. To illustrate the performance of the method and confirm the theoretical estimates, a selection of 2D and 3D numerical tests is provided.

Keywords: Weighted quadrature; Isogeometric analysis; Hierarchical B-splines

1. Introduction

Local and adaptive mesh refinement methods in isogeometric analysis have gained a notable attention in the last years and their mathematical theory has recently been established, see [1] and references therein. One of the more prominent tool in this context is provided by hierarchical B-spline construction [2–4]. The attractive advantage of the hierarchical spline model comes from a good balance between sound theoretical foundations, flexibility, and ease of implementation. A local refinement step is governed by simple conditions that activate/deactivate basis functions from hierarchically nested sequence of spline spaces. The use of the hierarchical approach in isogeometric analysis was originally proposed in [2] and subsequently investigated in different directions, which range from the theory of adaptive methods [5–7] to engineering applications, see e.g., [8–10] and references therein.

The efficient formation of matrices in isogeometric Galerkin methods is a topic of active research. In this paper we focus on the weighted quadrature (WQ) approach, introduced in [11]. Other recent results and methods in this area are integration by interpolation and look-up [12,13], multiscale quadrature [14], sum-factorization [15,16],

* Corresponding author.
E-mail addresses: carlotta.giannelli@unifi.it (C. Giannelli), tadej.kanduc@fmf.uni-lj.si (T. Kanduč), martinelli@imati.cnr.it (M. Martinelli), giancarlo.sangalli@unipv.it (G. Sangalli), mattia.tani@imati.cnr.it (M. Tani).

https://doi.org/10.1016/j.cma.2022.115465
0045-7825/

the surrogate matrix method [17], reduced integration for FEM and immersed methods [18] and at superconvergent points [19], quadrature schemes for shell elements [20] and, beyond quadrature, the use of low-rank approximation [21] or GPUs [22].

The aim of WQ is to reduce the number of quadrature points that are needed to accurately compute integrals involving products of B-spline basis functions. In combination with sum-factorization and other implementational techniques, it reduces significantly the cost of formation of isogeometric matrices. The idea of WQ is that the test function plays the role of weight function in the integration, and therefore the quadrature weights depends on the test function. The advantage of this construction is that the number of exactness conditions to be imposed is less than for Gaussian quadrature, generalized Gaussian quadrature [23–26] or reduced quadrature [27–29]. Therefore, WQ requires less quadrature points and its number mildly depends on the spline degree. The extension of the weighted quadrature scheme to accurately integrate the elements of the stiffness matrix in linear elasticity was presented in [30].

In this paper we extend WQ to hierarchical B-splines with maximum regularity. Since the construction of the hierarchical basis is simply based on a suitable selection of standard B-splines at different levels of details, we can define hierarchical WQ as a linear combination of standard WQ on different tensor product levels. The proposed algorithm has a computational cost proportional to the number of degrees of freedom and advantageous properties with increasing spline degree. After discussing in detail the case of the mass matrix, we briefly explain the quadrature formulas for second order operators. To illustrate the performance of the method and confirm the theoretical estimates, a selection of 2D and 3D numerical tests for L^2-projection and linear elasticity is provided. Note that other types of matrices and PDE problems could conceptually be dealt in a similarly way, see [11]. Although we focus only on spline spaces with maximum regularity, this is not a restriction of WQ, see [30] for a detailed analysis on this case.

The structure of the paper is as follows. Preliminaries on hierarchical B-splines and weighted quadrature are recalled in Section 2. The WQ and its use in the matrix formation for hierarchical B-splines is then presented in Section 3, while the computational cost is studied in Section 4. Section 5 illustrates the numerical experiments and, finally, Section 6 concludes the paper.

2. Preliminaries

2.1. Hierarchical B-splines

We consider a nested sequence of $L+1$ multivariate tensor product spline spaces V^ℓ, for $\ell = 0, \dots, L$, of fixed degree p in any coordinate direction defined on a bounded closed hyper-rectangle $\Omega \subset \mathbb{R}^d$. By focusing on dyadic mesh refinement, we assume the spline spaces is defined on a sequence of suitably refined knot vectors so that $V^\ell \subset V^{\ell+1}$, for $\ell = 0, \dots, L-1$. It should be noted however that the hierarchical B-spline model can be considered also in connection with more general (non-uniform) mesh refinement rules, where each mesh element is subdivided in an arbitrary number of children elements. Moreover, not only h-refinement but also p-refinement can be combined with the construction of the spline hierarchy as long as the spaces remain nested between each pair $(\ell, \ell+1)$ of consecutive levels, for $\ell = 0, \dots, L-1$. The considered choice in the paper is dyadic (uniform) refinement and fixed spline degree at all levels, which is the standard setting for adaptive isogeometric methods, based on hierarchical B-splines, and a suitable compromise between accuracy and efficiency for related application algorithms. Note that the design and development of fast assembly and efficient numerical integration rules tailored on hierarchical B-spline constructions are key ingredients for the subsequent development of more flexible adaptive approximation schemes.

In direction k of the domain Ω, the level ℓ basis $\mathcal{B}_k^\ell$ consists of $N_{\mathcal{B}_k}^\ell$ univariate B-splines b_{k,i_k}^ℓ,

$$\mathcal{B}_k^\ell = \left\{ b_{k,i_k}^\ell \ : \ i_k = 1, 2, \dots, N_{\mathcal{B}_k}^\ell \right\}.$$

The multivariate spline space V^ℓ on Ω can be defined as the span of the tensor product B-spline basis functions $b_{\boldsymbol{i}}^\ell$,

$$\mathcal{B}^\ell := \left\{ b_{\boldsymbol{i}}^\ell := \prod_{k=1}^{d} b_{k,i_k}^\ell \ : \ b_{k,i_k}^\ell \in \mathcal{B}_k^\ell \right\}$$

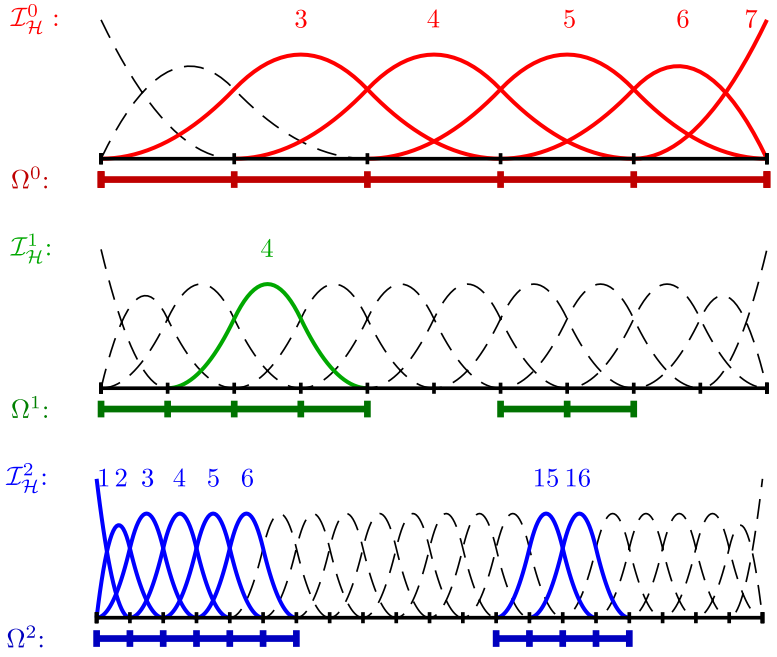

Fig. 1. An example of an univariate hierarchical space. Mesh cells and B-splines for levels 0,1,2 are depicted in top, middle and bottom plot, respectively. The hierarchical space is determined by domains Ω^ℓ, its basis functions in $\mathcal{I}^\ell_{\mathcal{H}}$ are marked and labeled in each plot.

with respect to the index set

$$\mathcal{I}^\ell_{\mathcal{B}} := \left\{ \boldsymbol{i} := (i_1, \dots, i_d) \in \mathbb{N}^d \ : \ 1 \leq i_k \leq N^\ell_{\mathcal{B}_k},\ k = 1, \dots, d \right\}.$$

We denote the rectilinear mesh grid of level ℓ by $\mathcal{M}^\ell$. To localize the refinement regions at different hierarchical levels, we also consider a nested sequence of closed subsets of $\Omega^0 := \Omega$ given by

$$\Omega^0 \supseteq \Omega^1 \supseteq \ldots \supseteq \Omega^{L+1} = \emptyset.$$

The hierarchical mesh $\mathcal{M}$ collects the grid elements $M \in \mathcal{M}^\ell$, which are not included in any refined region Ω^m of higher level $m > \ell$,

$$\mathcal{M} := \left\{ M \in \mathcal{M}^\ell \ : \ M \subseteq \Omega^\ell, M \not\subseteq \Omega^{\ell+1},\ \ell = 0, \dots, L \right\}.$$

We define the hierarchical B-spline basis [2] with respect to the hierarchical mesh $\mathcal{M}$ as

$$\mathcal{H}(\mathcal{M}) := \left\{ b^\ell_{\boldsymbol{i}} \in \mathcal{B}^\ell \ : \ \boldsymbol{i} \in \mathcal{I}^\ell_{\mathcal{H}},\ \ell = 0, \dots, L \right\},$$

where

$$\mathcal{I}^\ell_{\mathcal{H}} := \left\{ \boldsymbol{i} \in \mathcal{I}^\ell_{\mathcal{B}} \ : \ \mathrm{supp}(b^\ell_{\boldsymbol{i}}) \subset \Omega^\ell, \mathrm{supp}(b^\ell_{\boldsymbol{i}}) \not\subset \Omega^{\ell+1} \right\}.$$

Note that in this paper we consider the support of a function as an open set. The introduced empty set Ω^{L+1} ensures that all B-splines of the finest level L whose support is fully contained in Ω^L through the index set $\mathcal{I}^L_{\mathcal{H}}$ are activated.

Each basis function $b^\ell_{\boldsymbol{i}} \in \mathcal{H}$ is uniquely identified by its level ℓ and by the multi-index $\boldsymbol{i} \in \mathcal{I}^\ell_{\mathcal{H}}$. Hence we can define the set of basis identifiers

$$\mathcal{I}_{\mathcal{H}} := \bigcup_{\ell=0}^{L} \{(\ell, \boldsymbol{i}) \ : \ \boldsymbol{i} \in \mathcal{I}^\ell_{\mathcal{H}}\}. \tag{1}$$

The cardinality of $\mathcal{I}_{\mathcal{H}}$ is denoted by $N_{\mathcal{H}}$. Hierarchical B-splines are non-negative, linear independent, and allow localized mesh refinement by suitably selecting basis functions with a varying level of resolution. An univariate example of a hierarchical basis is depicted in Fig. 1.

In order to limit the interaction between B-splines introduced at very different levels of the spline hierarchy, we consider *admissible* meshes. A hierarchical mesh $\mathcal{M}$ is admissible of class r, with $2 \leq r < L+1$, if the hierarchical B-splines taking non-zero values on any element $Q \in \mathcal{M}$ belong to at most r successive levels. We refer to [1,5] for more details concerning admissible meshes and their properties, and to [31] for the presentation of the refinement algorithms which guarantee the construction of hierarchical mesh configurations with different class of admissibility. Note that the suitably graded meshes generated via these algorithms are characterized by a different (stronger) version of admissibility, which is easier to obtain via automatically-driven refinement rules.

2.2. WQ for tensor product splines

The goal of WQ is to reduce the number of quadrature points in the computation of Galerkin integrals for smooth B-spline basis functions. In combination with the sum-factorization and replacement of the element-wise assembly loop by a direct function-wise calculation of the matrix entries, the cost for the formation of isogeometric Galerkin matrices goes from $O(p^{3d}N)$ FLOPS down to $O(p^{d+1}N)$ FLOPS, where p and N denote the spline degree and number of degrees of freedom, respectively.

Consider $\mathcal{B}^\ell$, the level ℓ tensor product B-spline basis, then we can directly apply the construction of [11] and introduce for each $\boldsymbol{i} \in \mathcal{I}_{\mathcal{B}}^\ell$, the following WQ

$$\bar{\Omega}_{\boldsymbol{i}}^\ell(v) := \sum_{\mathbf{q} \in \mathcal{I}_{\mathcal{Q}}^\ell} \bar{w}_{\boldsymbol{i},\mathbf{q}}^\ell v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^\ell\right) \approx \int_{[0,1]^d} v(\boldsymbol{x}) b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) d\boldsymbol{x} \tag{2}$$

where $b_{\boldsymbol{i}}^\ell \in \mathcal{B}^\ell$, $\bar{w}_{\boldsymbol{i},\mathbf{q}}^\ell$ are the quadrature weights and $\bar{\boldsymbol{x}}_{\mathbf{q}}^\ell$ the quadrature points. For a better distinction between tensor product and hierarchical objects, we use bars on the top of symbols for quadrature rules, points and weights in the former case. Note that weights depend on $\boldsymbol{i}$, that is, on $b_{\boldsymbol{i}}^\ell$, which plays the role of a weight function for the integration. The rule (2) can be used to approximate the $(\boldsymbol{i}, \boldsymbol{j})$-entry of the mass matrix at level ℓ, indeed

$$\int_{[0,1]^d} c(\boldsymbol{x}) b_{\boldsymbol{j}}^\ell(\boldsymbol{x}) b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) d\boldsymbol{x} \approx \bar{\Omega}_{\boldsymbol{i}}^\ell(c b_{\boldsymbol{j}}^\ell), \tag{3}$$

where the given function c takes into account the determinant of the Jacobian of the parametrization map. More generally, any matrix arising in an isogeometric Galerkin method can be formed by suitable WQ, see [11]. The accuracy of WQ is related to the exactness conditions that the rule satisfies. In the case of (2)–(3), a typical request is

$$\bar{\Omega}_{\boldsymbol{i}}^\ell(b_{\boldsymbol{j}}^\ell) = \int_{[0,1]^d} b_{\boldsymbol{j}}^\ell(\boldsymbol{x}) b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) d\boldsymbol{x}, \qquad b_{\boldsymbol{j}}^\ell \in \mathcal{B}^\ell. \tag{4}$$

Though not necessary, following [11], the quadrature points $\bar{\boldsymbol{x}}_{\mathbf{q}}^\ell$ in (2) do not depend on $\boldsymbol{i}$. The set of d-variate quadrature points is defined as the following tensor product

$$\mathcal{Q}^\ell := \mathcal{Q}_1^\ell \times \cdots \times \mathcal{Q}_d^\ell\,, \tag{5}$$

where $\mathcal{Q}_k^\ell := \{\bar{x}_{k,q_k}^\ell\}_{q_k=1}^{R_k^\ell}$ is the set of univariate quadrature points in the kth direction and R_k^ℓ is the number of quadrature points along that direction. In the case of splines of maximum regularity, these points can be selected as the midpoints and endpoints of the knot spans, with the exception of the first and last knot spans where we can select $p+1$ uniformly distributed points, see [11]. We also introduce the tensor product set of multi-indices, associated to $\mathcal{Q}^\ell$ as

$$\mathcal{I}_{\mathcal{Q}}^\ell := \{\mathbf{q} = (q_1, \ldots, q_d) \in \mathbb{N}^d \ : \ 1 \leq q_k \leq R_k^\ell\,,\ 1 \leq k \leq d\}\,.$$

Even though the quadrature points are defined globally, only those in the support of $b_{\boldsymbol{i}}^\ell$ are active for $\bar{\Omega}_{\boldsymbol{i}}^\ell$. Thus, the active quadrature points of $\bar{\Omega}_{\boldsymbol{i}}^\ell$ depend on $\boldsymbol{i}$. For simpler explanation purposes this is formalized by setting to zero the weights $\{\bar{w}_{\boldsymbol{i},\mathbf{q}}^\ell\}_{\mathbf{q} \in \mathcal{I}_{\mathcal{Q}}^\ell}$ that correspond to points outside the support of $b_{\boldsymbol{i}}^\ell$ (in our implementation, however, only point indices with non-zero weights are stored in the data structures to be more efficient). The non-zero weights are computed by imposing the univariate local exactness conditions, leading to linear problems whose solution cost is not prevailing in the overall matrix formation cost.

More generally, WQ can be used for any second-order linear operator on a tensor product splines, see [11]. When restricted to splines of level ℓ, the entries of the stiffness matrix associated to such operator have the form

$$\sum_{\beta,\gamma=1}^{d} \int_{[0,1]^d} a_{\beta\gamma}(\boldsymbol{x})\partial_\beta b_{\boldsymbol{i}}^\ell(\boldsymbol{x})\partial_\gamma b_{\boldsymbol{j}}^\ell(\boldsymbol{x})d\boldsymbol{x} \; + \; \sum_{\gamma=1}^{d} \int_{[0,1]^d} b_\gamma(\boldsymbol{x}) b_{\boldsymbol{i}}^\ell(\boldsymbol{x})\partial_\gamma b_{\boldsymbol{j}}^\ell(\boldsymbol{x})d\boldsymbol{x} + \\ + \int_{[0,1]^d} c(\boldsymbol{x}) b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) b_{\boldsymbol{j}}^\ell(\boldsymbol{x})d\boldsymbol{x}, \qquad \boldsymbol{i}, \boldsymbol{j} \in \mathcal{I}_{\mathcal{B}}^\ell, \tag{6}$$

where, similarly as before, the functions $a_{\beta\gamma}$, b_γ and c take into account the geometry mapping and the coefficients of the operator. For a fixed $\boldsymbol{i} \in \mathcal{I}_{\mathcal{B}}^\ell$, each of the integrals in the above expression can be approximated independently using a different weighted quadrature rule, precisely

$$\bar{\Omega}_{\boldsymbol{i}}^{\ell,\beta,\gamma}(v) \coloneqq \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^\ell} \bar{w}_{\boldsymbol{i},\mathbf{q}}^{\ell,\beta,\gamma} v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^\ell\right) \approx \int_{[0,1]^d} \partial_\beta b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) v(\boldsymbol{x})d\boldsymbol{x}, \\ \bar{\Omega}_{\boldsymbol{i}}^{\ell,\gamma}(v) \coloneqq \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^\ell} \bar{w}_{\boldsymbol{i},\mathbf{q}}^{\ell,\gamma} v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^\ell\right) \approx \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) v(\boldsymbol{x})d\boldsymbol{x}, \tag{7}$$

and the rule $\bar{\Omega}_{\boldsymbol{i}}^\ell$ already introduced for the mass matrix. Note that the index β is associated with derivatives that appears in the integral measure, while the index γ is associated with derivatives for which the rule is exact. Moreover, the set of quadrature points is the same as for the mass matrix (in particular, we still need just 2^d points per element if not close to a boundary of the domain).

A typical choice for the exactness conditions characterizing these rules is

$$\bar{\Omega}_{\boldsymbol{i}}^{\ell,\beta,\gamma}(\partial_\gamma b_{\boldsymbol{j}}^\ell) = \int_{[0,1]^d} \partial_\beta b_{\boldsymbol{i}}^\ell(\boldsymbol{x})\partial_\gamma b_{\boldsymbol{j}}^\ell(\boldsymbol{x})d\boldsymbol{x}, \\ \bar{\Omega}_{\boldsymbol{i}}^{\ell,\gamma}(\partial_\gamma b_{\boldsymbol{j}}^\ell) = \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x})\partial_\gamma b_{\boldsymbol{j}}^\ell(\boldsymbol{x})d\boldsymbol{x}, \qquad b_{\boldsymbol{j}}^\ell \in \mathcal{B}^\ell. \tag{8}$$

An alternative approach [30] is not considered in this study; it is considered less efficient in our setting since it depends on more quadrature points, despite relying on fewer number of quadrature rules.

3. WQ and matrix formation for hierarchical B-splines

3.1. Definition of WQ for mass matrix

A weighted quadrature rule, associated to an active basis function $b_{\boldsymbol{i}}^\ell \in \mathcal{H}$, is denoted by $\Omega_{\boldsymbol{i}}^\ell$. Its quadrature points and weights are jointly indexed with respect to an index set that is denoted by $\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}$. Namely, let

$$\mathcal{Q}^{(\ell,\boldsymbol{i})} \coloneqq \left\{\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} \tag{9}$$

be a set of quadrature points and the set of the corresponding weights is $\left\{w_{\boldsymbol{i},\mathbf{q}}^\ell\right\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}}$. The quadrature rule $\Omega_{\boldsymbol{i}}^\ell$ applied to an auxiliary function v has the following form

$$\Omega_{\boldsymbol{i}}^\ell(v) \coloneqq \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^\ell v\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right) \approx \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) v(\boldsymbol{x})d\boldsymbol{x}. \tag{10}$$

A distinguishing feature of this structure is that both the set of quadrature points and the set of quadrature weights depend on the considered test function $b_{\boldsymbol{i}}^\ell$. However, as we will see in the following, $\mathcal{Q}^{(\ell,\boldsymbol{i})}$ can be conveniently selected as a subset of a global tensor product grid, which is chosen a priori.

Similarly as in the non-hierarchical case discussed in the previous section, the quadrature rules are characterized by exactness conditions. More specifically, we require that the rules are exact for all functions in the spline space, or equivalently that

$$\Omega_{\boldsymbol{i}}^\ell(b_{\boldsymbol{j}}^m) = \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^\ell \, b_{\boldsymbol{j}}^m\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right) = \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) b_{\boldsymbol{j}}^m(\boldsymbol{x})d\boldsymbol{x}, \qquad (m,\boldsymbol{j}) \in \mathcal{I}_{\mathcal{H}}. \tag{11}$$

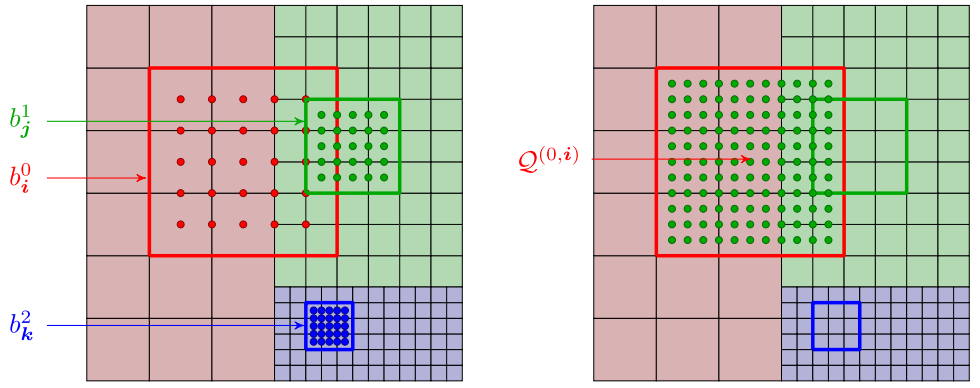

Fig. 2. Example of a hierarchical mesh with three different levels. Three basis functions from $\mathcal{B}^\ell$, with $\ell = 0, 1, 2$ are considered. The supports and quadrature points for these B-splines from spaces $\mathcal{B}^0$ (red), $\mathcal{B}^1$ (green) and $\mathcal{B}^2$ (blue) are depicted on the hierarchical mesh (left). The quadrature points for $b_i^0 \in \mathcal{H}$ used in weighted quadrature for hierarchical B-splines are also shown (right). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For a given pair $(\ell, \boldsymbol{i}) \in \mathcal{I}_\mathcal{H}$ we define $\nu(\ell, \boldsymbol{i})$ as the finest hierarchical level whose functions interacts with $b_{\boldsymbol{i}}^\ell$, i.e.,

$$\nu(\ell, \boldsymbol{i}) := \max \left\{ m \ : \ \exists\, b_{\boldsymbol{j}}^m \in \mathcal{H} \text{ s.t. } \operatorname{supp}(b_{\boldsymbol{i}}^\ell) \cap \operatorname{supp}(b_{\boldsymbol{j}}^m) \neq \emptyset \right\}. \tag{12}$$

To make the notation lighter, the argument $(\ell, \boldsymbol{i})$ in $\nu(\ell, \boldsymbol{i})$ will be sometimes omitted, since the dependence on the basis identifier will be clear from the context.

Any active basis function $b_{\boldsymbol{j}}^m$ that interacts with $b_{\boldsymbol{i}}^\ell$ (including itself) can be written as a linear combination of basis functions of level $\nu(\ell, \boldsymbol{i})$, that is

$$b_{\boldsymbol{j}}^m = \sum_{\boldsymbol{t} \in \mathcal{I}_\mathcal{B}^\nu} \alpha_{\boldsymbol{j},\boldsymbol{t}}^{m,\nu}\, b_{\boldsymbol{t}}^\nu \qquad \forall\, (m, \boldsymbol{j}) \in \mathcal{I}_\mathcal{H} \quad \text{such that} \quad \operatorname{supp}(b_{\boldsymbol{i}}^\ell) \cap \operatorname{supp}(b_{\boldsymbol{j}}^m) \neq \emptyset, \tag{13}$$

with $\alpha_{\boldsymbol{j},\boldsymbol{t}}^{m,\nu} > 0$ if $\operatorname{supp}(b_{\boldsymbol{t}}^\nu) \subseteq \operatorname{supp}(b_{\boldsymbol{j}}^m)$ and $\alpha_{\boldsymbol{j},\boldsymbol{t}}^{m,\nu} = 0$ otherwise.

In order to define the quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^\ell$ for the hierarchical space $\mathcal{H}$, we rely on the definitions and relations introduced in the previous section for tensor product spaces. In (10) we take

$$\mathcal{I}_\mathcal{Q}^{(\ell,\boldsymbol{i})} := \left\{ \mathbf{q} \in \mathcal{I}_\mathcal{Q}^\nu \ : \ \bar{\boldsymbol{x}}_{\mathbf{q}}^\nu \in \mathcal{Q}^\nu \cap \operatorname{supp}(b_{\boldsymbol{i}}^\ell) \right\}$$

and let the quadrature points be $\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell = \bar{\boldsymbol{x}}_{\mathbf{q}}^\nu$ for every $\mathbf{q} \in \mathcal{I}_\mathcal{Q}^{(\ell,\boldsymbol{i})}$, hence (from definition (9)) we have

$$\mathcal{Q}^{(\ell,\boldsymbol{i})} = \mathcal{Q}^\nu \cap \operatorname{supp}(b_{\boldsymbol{i}}^\ell). \tag{14}$$

See Fig. 2 for an example of quadrature points for a basis function $b_{\boldsymbol{i}}^0 \in \mathcal{H}$. Since it interacts with a level 1 basis function and not with a level 2 one, it inherits a local set of level 1 quadrature points $\bar{\boldsymbol{x}}_{\mathbf{q}}^1$.

The quadrature weights for $b_{\boldsymbol{i}}^\ell$ are simply

$$w_{\boldsymbol{i},\mathbf{q}}^\ell = \sum_{\boldsymbol{s} \in \mathcal{I}_\mathcal{B}^\nu} \alpha_{\boldsymbol{i},\boldsymbol{s}}^{\ell,\nu}\, \bar{w}_{\boldsymbol{s},\mathbf{q}}^\nu \tag{15}$$

for every $\mathbf{q} \in \mathcal{I}_\mathcal{Q}^{(\ell,\boldsymbol{i})}$, and the coefficients $\alpha_{\boldsymbol{i},\boldsymbol{s}}^{\ell,\nu}$ are the expansion coefficients of $b_{\boldsymbol{i}}^\ell$ on the basis $\mathcal{B}^\nu$ as in (13).

In the following proposition we show that this choice for the quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^\ell$ satisfies the imposed exactness conditions on the hierarchical space.

Proposition 1.

For $(\ell, \boldsymbol{i}) \in \mathcal{I}_\mathcal{H}$ the quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^\ell$ satisfies the exactness conditions (11) on the hierarchical space.

Proof. Let $(m, \boldsymbol{j}) \in \mathcal{I}_{\mathcal{H}}$. If $\mathrm{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \mathrm{supp}(b_{\boldsymbol{j}}^{m}) = \emptyset$, then the equation in (11) is trivially satisfied since the quadrature points $\mathcal{Q}^{(\ell,\boldsymbol{i})}$ belong to the support of $b_{\boldsymbol{i}}^{\ell}$. On the other hand, if $\mathrm{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \mathrm{supp}(b_{\boldsymbol{j}}^{m}) \neq \emptyset$, then

$$
\begin{aligned}
\mathfrak{Q}_{\boldsymbol{i}}^{\ell}\left(b_{\boldsymbol{j}}^{m}\right) &= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^{\ell} b_{\boldsymbol{j}}^{m}\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^{\ell}\right) \\
&= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^{\ell} b_{\boldsymbol{j}}^{m}\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right) \\
&= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{\nu}} \left(\sum_{s\in\mathcal{I}_{\mathcal{B}}^{n}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \bar{w}_{s,\mathbf{q}}^{\nu}\right) \left(\sum_{t\in\mathcal{I}_{\mathcal{B}}^{\nu}} \alpha_{\boldsymbol{j},t}^{m,\nu} b_{t}^{\nu}\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right)\right) \\
&= \sum_{s,t\in\mathcal{I}_{\mathcal{B}}^{\nu}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \alpha_{\boldsymbol{j},t}^{m,\nu} \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{\nu}} \bar{w}_{s,\mathbf{q}}^{\nu} b_{t}^{\nu}\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right) \\
&= \sum_{s,t\in\mathcal{I}_{\mathcal{B}}^{\nu}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \alpha_{\boldsymbol{j},t}^{m,\nu} \int_{[0,1]^d} b_{s}^{\nu}(\boldsymbol{x}) b_{t}^{\nu}(\boldsymbol{x}) d\boldsymbol{x} \qquad (16) \\
&= \int_{[0,1]^d} \left(\sum_{s\in\mathcal{I}_{\mathcal{B}}^{\nu}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} b_{s}^{\nu}(\boldsymbol{x})\right) \left(\sum_{s\in\mathcal{I}_{\mathcal{B}}^{\nu}} \alpha_{\boldsymbol{j},t}^{m,\nu} b_{t}^{\nu}(\boldsymbol{x})\right) d\boldsymbol{x} \\
&= \int_{[0,1]^d} b_{\boldsymbol{i}}^{\ell}(\boldsymbol{x}) b_{\boldsymbol{j}}^{m}(\boldsymbol{x}) d\boldsymbol{x}.
\end{aligned}
$$

In (16) we use the property that for each $\boldsymbol{s} \in \mathcal{I}_{\mathcal{B}}^{\nu}$ the set of points $\mathcal{Q}^{\nu}$ and the set of weights $\left\{\bar{w}_{s,\mathbf{q}}^{\nu}\right\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{\nu}}$ satisfy the exactness conditions (4) on the level $\nu = \nu(\ell, \boldsymbol{i})$. □

Remark 1. Quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^{\ell}$ in (10) of level ℓ is actually a linear combination of quadrature rules defined at level ν. Namely, if $\mathrm{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \mathrm{supp}(v) \neq \emptyset$ we can define $\mathcal{I}^{(\ell,\boldsymbol{i})} := \{\boldsymbol{s} \in \mathcal{I}_{\mathcal{B}}^{\nu} \ : \ \mathrm{supp}(b_{\boldsymbol{s}}^{\nu}) \subseteq \mathrm{supp}(b_{\boldsymbol{i}}^{\ell})\}$ and derive

$$
\begin{aligned}
\mathfrak{Q}_{\boldsymbol{i}}^{\ell}(v) &= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^{\ell} v\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^{\ell}\right) \\
&= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} w_{\boldsymbol{i},\mathbf{q}}^{\ell} v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right) \\
&= \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,\boldsymbol{i})}} \sum_{s\in\mathcal{I}^{(\ell,\boldsymbol{i})}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \bar{w}_{s,\mathbf{q}}^{\nu} v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right) \\
&= \sum_{s\in\mathcal{I}^{(\ell,\boldsymbol{i})}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\nu,s)}} \bar{w}_{s,\mathbf{q}}^{\nu} v\left(\bar{\boldsymbol{x}}_{\mathbf{q}}^{\nu}\right) \\
&= \sum_{s\in\mathcal{I}^{(\ell,\boldsymbol{i})}} \alpha_{\boldsymbol{i},s}^{\ell,\nu} \bar{\mathfrak{Q}}_{s}^{\nu}(v).
\end{aligned}
$$

In particular, the quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^{\ell}$ is determined by those rules whose support is included in the support of $b_{\boldsymbol{i}}^{\ell}$. Note that a construction of weighted quadrature rules as linear combination of other rules was recently addressed also in the context of (trimmed) tensor product patches [32].

Remark 2. It might seem intuitive that the quadrature points relative to a coarse basis function should be taken on a finer level only in the overlap with the support of finer basis function. In fact, the present version of weighted quadrature requires that all points associated to a function are taken on the finest level it interacts with. The reason behind this is twofold. First, each test function is associated with a single quadrature rule, i.e. a single set of points and weights, which is used regardless of the level of the interacting trial function. We have numerically tested the use of different rules for the same test function, taking the points and weights from the finest level between the test

and the trial function, but we have observed a loss of the order of convergence. Second, the exactness conditions used to compute the rules involve functions that belong to the same hierarchical level, similarly as in [11]. This means in particular that there is no guarantee that "mixing" rules of different levels in the support of a given test function (i.e. using fine points/weights in the overlap with fine functions and coarse points/weights in the overlap with coarse functions) will result in a rule that satisfies the exactness conditions (11).

3.2. Preprocessing: computing the quadrature points and weights

Since the quadrature weights are not known in advance for every possible mesh, degree, level and interaction, they need to be computed efficiently in the preprocessing phase, before utilizing them in the matrix formation phase. For computational efficiency, we fully exploit the tensor product structure of the active basis functions $b_{\boldsymbol{i}}^{\ell}$ and of the quadrature points. Quadrature weights are therefore obtained in two steps. First, we compute the univariate quadrature weights of level ν, defined in (12), by solving the linear systems arising from the univariate exactness conditions analogous to (4); this is the same as in [11]. Then, the univariate quadrature weights for the WQ associated to an active basis function $b_{\boldsymbol{i}}^{\ell}$ are computed as linear combination of level ν quadrature weights, analogously to (15).

The quadrature points and weights for d-variate B-splines are stored and used as d-tuples of the univariate points and univariate weights, respectively, in order to be ready for the sum-factorization used in the matrix formation.

To avoid redundant computations, all the active basis functions are clustered with respect to the value of ν so that the univariate routines are engaged only once for each level, i.e., we classify the basis functions of $\mathcal{H}$ with respect to ν by defining the *sets of level n interacting functions*

$$F^n := \{b_{\boldsymbol{i}}^{\ell} \ : \ (\ell, \boldsymbol{i}) \in \mathcal{I}_{\mathcal{H}} \ , \ \nu(\ell, \boldsymbol{i}) = n\}.$$

It is trivial to check the following properties:

$$\mathcal{H} = \bigcup_{n \leq L} F^n, \tag{17}$$

$$F^m \cap F^n = \emptyset \text{ if } m \neq n, \tag{18}$$

$$F^n \text{ only contains functions of } \mathcal{H} \text{ of levels } \leq n \ . \tag{19}$$

The classification of the active basis function $b_{\boldsymbol{i}}^{\ell}$ with respect to the maximum level of interaction $\nu(\ell, \boldsymbol{i})$ is described in Algorithm 1.

Algorithm 1: `classify_basis_functions`

Input : $\mathcal{H}$

Initialize $F^n = \emptyset$ for $n = 0, \dots, L$
foreach $(\ell, \boldsymbol{i}) \in \mathcal{I}_{\mathcal{H}}$ **do**
  $F^{\nu(\ell,\boldsymbol{i})} = F^{\nu(\ell,\boldsymbol{i})} \cup b_{\boldsymbol{i}}^{\ell}$
end foreach

Output: $\{F^n\}_{n \leq L}$

Because of (13) and (19), every function $b_{\boldsymbol{i}}^{\ell} \in F^n$ can be written as linear combination of functions $b_{\boldsymbol{j}}^{n}$ from level n, and a similar formula holds for its quadrature weights (see (15)). Analogously, the same can be said for the (univariate) components: b_{k,i_k}^{ℓ} can be written as a linear combination of level n functions b_{k,j_k}^{n}, and w_{k,i_k,q_k}^{ℓ} as a linear combination of $\bar{w}_{k,j_k,q_k}^{n}$,

$$b_{k,i_k}^{\ell} = \sum_{j_k \in D_{k,i_k}^{\ell,n}} \alpha_{k,i_k,j_k}^{\ell,n} \, b_{k,j_k}^{n}, \qquad w_{k,i_k,q_k}^{\ell} = \sum_{j_k \in D_{k,i_k}^{\ell,n}} \alpha_{k,i_k,j_k}^{\ell,n} \bar{w}_{k,j_k,q_k}^{n} \tag{20}$$

for $k = 1, 2, \dots, d$, where

$$D_{k,i_k}^{\ell,n} = \{j_k \in \{1, \dots, N_{\mathcal{B}_k}^{n}\} \ : \ \text{supp}(b_{k,j_k}^{n}) \subseteq \text{supp}(b_{k,i_k}^{\ell})\}.$$

To switch from the d-variate to the univariate setting, we first need to define two auxiliary functions π_k and τ_k,

$$\pi_k \boldsymbol{i} := i_k, \qquad \tau_k \boldsymbol{i} := (i_1, \dots, i_k),$$

acting on a multi-index $\boldsymbol{i} = (i_1, \dots, i_d)$. Then, for $k = 1, \dots, d$ we introduce the set of indices of univariate B-spline that are used to define the functions $b_{\boldsymbol{i}}^{\ell} \in \mathcal{H}^{\ell} \cap F^n$:

$$G_k^{\ell,n} := \{i_k = \pi_k \boldsymbol{i} \ : \ b_{\boldsymbol{i}}^{\ell} \in \mathcal{H}^{\ell} \cap F^n\}. \tag{21}$$

Finally, we define

$$D_k^n := \bigcup_{\ell=0}^{n} \bigcup_{i_k \in G_k^{\ell,n}} D_{k,i_k}^{\ell,n}. \tag{22}$$

The sets D_k^n identify the univariate quadrature weights that are needed to set up the WQ rules for functions of F^n along the $k-$th direction. Precisely, in the next step for each $j_k \in D_k^n$ we compute the non-zero univariate quadrature weights $\{\bar{w}_{k,j_k,q_k}^n\}_{q_k=1}^{R_k^n}$ associated to $b_{k,j_k}^n \in \mathcal{B}_k^n$ by imposing the univariate exactness conditions analogous to (4), exactly as done in [11]. Namely, we impose that

$$\bar{w}_{k,j_k,q_k}^n = 0 \qquad \text{if } \bar{x}_{k,q_k}^n \notin \operatorname{supp}(b_{k,j_k}^n),$$

while non-zero quadrature weights are obtained by solving the linear system

$$\sum_{q_k \in A_{k,j_k}^{n,n}} \bar{w}_{k,j_k,q_k}^n b_{k,t_k}^n(\bar{x}_{k,q_k}^n) = \int_0^1 b_{k,j_k}^n(\xi)\, b_{k,t_k}^n(\xi)\, d\xi, \qquad b_{k,t_k}^n \in B_{k,j_k}^n, \tag{23}$$

where

$$A_{k,j_k}^{\ell,n} := \{q_k \in \{1, \dots, R_k^n\} \ : \ \bar{x}_{k,q_k}^n \in \mathcal{Q}_k^n \cap \operatorname{supp}(b_{k,j_k}^{\ell})\}, \tag{24}$$

$$B_{k,j_k}^n := \{b_{k,t_k}^n \in \mathcal{B}_k^n \ : \ \operatorname{supp}(b_{k,j_k}^n) \cap \operatorname{supp}(b_{k,t_k}^n) \neq \emptyset\} \tag{25}$$

are the set of indices of quadrature points inside the support of the basis function b_{k,j_k}^{ℓ}, and the corresponding interacting trial univariate functions, respectively. Note that if the support of b_{k,j_k}^{ℓ} does not include the first or last element, the local system has exactly $2p+1$ unknowns and equations. In the case of a rectangular underdetermined system, which might happen if the support includes the first or last element, we compute the solution with minimum Euclidean norm. The construction of the univariate quadrature weights is summarized in Algorithm 2.

Algorithm 2: `compute_1Dweights`

Input : $\mathcal{B}_k^n$, j_k, $\mathcal{Q}_k^n$

compute the indices $A_{k,j_k}^{n,n}$ from (24)

find the interacting trial univariate functions B_{k,j_k}^n from (25)

compute non-zero weights $\{\bar{w}_{k,j_k,q_k}^n\}_{q_k \in A_{k,j_k}^{n,n}}$ by solving the linear system (23)

Output: $\{\bar{w}_{k,j_k,q_k}^n\}_{q_k \in A_{k,j_k}^{n,n}}$

Up to this point, we have defined the univariate quadrature points and computed the univariate quadrature weights, associated to all the basis functions in $\mathcal{B}_k^n$ that are needed to represent functions in F^n as linear combination of functions of level n, by using (13). Using (20) we can then compute the level ℓ univariate quadrature weights

$$\mathcal{W}_{k,i_k}^{\ell,n} := \left\{ w_{k,i_k,q_k}^{\ell} = \sum_{j_k \in D_{k,i_k}^{\ell,n}} \alpha_{k,i_k,j_k}^{\ell,n} \bar{w}_{k,j_k,q_k}^n \ : \ q_k \in A_{k,i_k}^{\ell,n} \right\} \qquad k = 1, \dots, d\ , \tag{26}$$

for each index $i_k \in G_k^{\ell,n}$.

The last preprocessing phase is to define the subset of d-dimensional quadrature points that are contained by the support of functions in F^n, that will be used in the matrix formation phase for the evaluation of the non-tensor product coefficients. The union of support of basis functions in F^n,

$$\Psi^n := \bigcup_{b_{\boldsymbol{i}}^{\ell} \in F^n} \operatorname{supp}(b_{\boldsymbol{i}}^{\ell}), \tag{27}$$

is a set of d-dimensional boxes in $[0,1]^d$ that can be described as a set of mesh cells on level n, which in general does not have a tensor product structure. The d-dimensional level n quadrature points are simply defined as

$$\mathcal{Q}^n_\Psi := \mathcal{Q}^n \cap \Psi^n. \tag{28}$$

Remark 3. Due to nestedness of quadrature points $\mathcal{Q}^n$ with respect to level n, there are configurations in which some points are defined in multiple levels. For the sake of efficiency, in our code we also store the union of all d dimensional set of points, $\mathcal{Q}_\Psi := \bigcup_{n\le L} \mathcal{Q}^n_\Psi$, that is used for the evaluation of the non tensor product coefficients.

The complete preprocessing phase is described by the Algorithm 3.

Algorithm 3: `preprocessing`

Input : $\mathcal{H}$

$\{F^n\}_{n=0}^L =$ `classify_basis_functions(`$\mathcal{H}$`)` (Alg. 1)
for $n = 0, \dots, L$ *such that* $F^n \neq \emptyset$ **do**
  for $k = 1, \dots, d$ **do**
    compute 1D quadrature points $\mathcal{Q}^n_k = \{\bar{x}^n_{k,q_k}\}_{q_k=1}^{R^n}$ for $\mathcal{B}^n_k$ (see Section 2.2)
    for $\ell = 0, \dots, n$ **do**
      compute $G^{\ell,n}_k$ from (21)
    end for
    compute D^n_k from (22)
    foreach $j_k \in D^n_k$ **do**
      $\{\bar{w}^n_{k,j_k,q_k}\}_{q_k \in A^{n,n}_{k,j_k}}$ `= compute_1Dweights(`$\mathcal{B}^n_k$, j_k, $\mathcal{Q}^n_k$`)` (Alg. 2)
    end foreach
    $\mathcal{W}^n_k = \emptyset$
    for $\ell = 0, \dots, n$ **do**
      foreach $i_k \in G^{\ell,n}_k$ **do**
        compute weights $\mathcal{W}^{\ell,n}_{k,i_k}$ from (26)
        $\mathcal{W}^n_k = \mathcal{W}^n_k \cup \mathcal{W}^{\ell,n}_{k,i_k}$
      end foreach
    end for
  end for
  compute the d-dimensional tensor product points $\mathcal{Q}^n$ from (5)
  compute Ψ^n from (27)
  compute $\mathcal{Q}^n_\Psi$ from (28)
end for

Output: $F^n, \mathcal{W}^n_1, \dots, \mathcal{W}^n_d, \mathcal{Q}^n, \mathcal{Q}^n_\Psi$, for $n = 0, \dots, L$

3.3. Mass matrix formation: algorithm

The rows and columns of the mass matrix are associated to the test and trial functions, respectively. In order to emphasize the hierarchical level of a given basis function, we use row (or column) multi-index basis identifiers as in (1). Therefore the single entry of the mass matrix is denoted as $[M]_{(\ell,\boldsymbol{i}),(m,\boldsymbol{j})}$ and is defined as:

$$[M]_{(\ell,\boldsymbol{i}),(m,\boldsymbol{j})} = \int_{[0,1]^d} c(\boldsymbol{x}) b^\ell_{\boldsymbol{i}}(\boldsymbol{x}) b^m_{\boldsymbol{j}}(\boldsymbol{x})\, d\boldsymbol{x}\,,$$

where the function $c\colon [0,1]^d \to \mathbb{R}$ incorporates the determinant of the Jacobian of the mapping between the parametric domain $[0,1]^d$ and the physical domain Ω, and in general it does not have a tensor product structure.

Recalling the quadrature rule definition (10), and the fact that $w^{\ell}_{\boldsymbol{i},\mathbf{q}} = 0$ for $\bar{\boldsymbol{x}}^n_{\mathbf{q}} \notin \mathrm{supp}(b^{\ell}_{\boldsymbol{i}})$, we can write

$$[M]_{(\ell,\boldsymbol{i}),(m,\boldsymbol{j})} \approx \mathfrak{Q}^{\ell}_{\boldsymbol{i}}(c\, b^m_{\boldsymbol{j}}) = \sum_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}} w^{\ell}_{\boldsymbol{i},\mathbf{q}} c(\bar{\boldsymbol{x}}^n_{\mathbf{q}}) b^m_{\boldsymbol{j}}(\bar{\boldsymbol{x}}^n_{\mathbf{q}})\, d\boldsymbol{x}, \tag{29}$$

where $n = \nu(\ell, \boldsymbol{i})$.

Using the sum-factorization approach, we exploit $w^n_{\boldsymbol{i},\mathbf{q}} = \prod_{k=1}^d w^n_{k,i_k,\mathrm{q}_k}$ and $b^m_{\boldsymbol{j}}(\bar{\boldsymbol{x}}^n_{\mathbf{q}}) = \prod_{k=1}^d b^m_{k,j_k}(\bar{x}^n_{k,\mathrm{q}_k})$ and write (29) in terms of nested sums:

$$\begin{aligned}\mathfrak{Q}^{\ell}_{\boldsymbol{i}}(c b^m_{\boldsymbol{j}}) &= \sum_{\mathrm{q}_1,\dots,\mathrm{q}_d} \prod_{k=1}^{d} \left(w^{\ell}_{k,i_k,\mathrm{q}_k} b^m_{k,j_k}(\bar{x}^n_{k,\mathrm{q}_k}) \right) c(\bar{x}^n_{1,\mathrm{q}_1},\dots,\bar{x}^n_{d,\mathrm{q}_d}) \\ &= \sum_{\mathrm{q}_d} w^{\ell}_{d,i_d,\mathrm{q}_d} b^m_{d,j_d}(\bar{x}^n_{d,\mathrm{q}_d}) \left(\sum_{\mathrm{q}_{d-1}} \cdots \sum_{\mathrm{q}_1} w^{\ell}_{1,i_1,\mathrm{q}_1} b^m_{1,j_1}(\bar{x}^n_{1,\mathrm{q}_1}) c(\bar{x}^n_{1,\mathrm{q}_1},\dots,\bar{x}^n_{d,\mathrm{q}_d}) \right)\end{aligned} \tag{30}$$

where, in the summations above, each running index q_k, for $k = 1,\dots,d$, belongs to the set

$$Q^{n,\ell,m}_{k,i_k,j_k} := \{\mathrm{q}_k \in \{1,\dots,R^n_k\} : \bar{x}^n_{k,\mathrm{q}_k} \in \mathrm{supp}(b^{\ell}_{k,i_k}) \cap \mathrm{supp}(b^m_{k,j_k})\}. \tag{31}$$

The sum-factorization algorithm in essence is a clever way to perform the nested sum (30), that sequentially performs the integration along the directions $k = 1,\dots,d$, considering for each k all pairs of indices (i_k, j_k) that identify the weight and trial function respectively. Details are presented in the remaining part of this subsection, where, for the sake of notation simplicity, we will systematically omit the set $Q^{n,\ell,m}_{k,i_k,j_k}$ for the running index q_k in the summations.

In (30) we note that coefficient $c(\boldsymbol{x})$ must be evaluated at the points of the quadrature rule of level n. Moreover, from (17)–(19) we know that the sets $\{F^n\}_{n=0}^L$ (excluding the empty sets) form a partition of the hierarchical basis $\mathcal{H}$. This suggests to construct the matrix starting from an outer loop over $\{F^n\}_{n=0}^L$, i.e., over the different levels of quadrature rules, then for a given level n, compute the determinant of the Jacobian at $\mathcal{Q}^n_{\Psi}$ (i.e., on the points that have non-empty intersection with the support of each basis function in F^n) and set the values to be zero for the points $\mathcal{Q}^n \setminus \mathcal{Q}^n_{\Psi}$.

The key point here is that the evaluation of the non-tensor product coefficient $c(\boldsymbol{x})$ may be a costly operation, so we want to evaluate it just for the involved quadrature points, i.e., for each $\mathbf{q} \in \mathcal{I}^n_{\mathcal{Q}}$ we set:

$$C^n_{\mathbf{q}} = C^n_{(\mathrm{q}_1,\dots,\mathrm{q}_d)} := \begin{cases} c(\bar{\boldsymbol{x}}^n_{\mathbf{q}}) & \text{if } \bar{\boldsymbol{x}}^n_{\mathbf{q}} \in \mathcal{Q}^n_{\Psi} \\ 0 & \text{otherwise} \end{cases}. \tag{32}$$

Given a quadrature level $n \in \{0,\dots,L\}$ such that $F^n \neq \emptyset$, we loop over $\ell, m \in \{0,\dots,n\}$ and compute the connectivity between the test functions of $\mathcal{H}^{\ell} \cap F^n$ and the trial functions of $\mathcal{H}^m$, i.e.,

$$K^n_{\ell,m} := \{(\boldsymbol{i},\boldsymbol{j}) \in \mathcal{I}^{\ell}_{\mathcal{B}} \times \mathcal{I}^m_{\mathcal{B}} \ :\ b^{\ell}_{\boldsymbol{i}} \in \mathcal{H}^{\ell} \cap F^n\,,\ b^m_{\boldsymbol{j}} \in \mathcal{H}^m\,,\ \mathrm{supp}(b^{\ell}_{\boldsymbol{i}}) \cap \mathrm{supp}(b^m_{\boldsymbol{j}}) \neq \emptyset\}. \tag{33}$$

At this point we can apply the sum-factorization algorithm that allows us to evaluate the mass-matrix entries.

The integration along direction $k = 1$ writes as

$$\begin{aligned} I^{(1)}_{(i_1),(j_1);(\mathrm{q}_2,\dots,\mathrm{q}_d)} &:= \sum_{\mathrm{q}_1} w^{\ell}_{1,i_1,\mathrm{q}_1} b^m_{1,j_1}(\bar{x}^n_{1,q_1}) C^n_{(\mathrm{q}_1,\dots,\mathrm{q}_d)} \\ &= \sum_{\mathrm{q}_1} w^{\ell}_{1,i_1,\mathrm{q}_1} b^m_{1,j_1}(\bar{x}^n_{1,q_1}) I^{(0)}_{(),();(\mathrm{q}_1,\dots,\mathrm{q}_d)}\,, \end{aligned} \tag{34}$$

where we have defined $I^{(0)}_{(),();(\mathrm{q}_1,\dots,\mathrm{q}_d)} := C^n_{(\mathrm{q}_1,\dots,\mathrm{q}_d)}$, which only depends on the d-tuple of indices associated to the quadrature points. Performing the summation over q_1 we have as result $I^{(1)}_{(i_1),(j_1);(\mathrm{q}_2,\dots,\mathrm{q}_d)}$ that depends on the pair (i_1, j_1) (related to the univariate test and trial basis along direction 1) and on the $(d-1)$-tuple $(\mathrm{q}_2,\dots,\mathrm{q}_d)$ (related to univariate quadrature points along the directions $2,\dots,d$). The integration along directions $k = 2,\dots,d-1$ then writes as:

$$I^{(k)}_{(i_1,\dots,i_k),(j_1,\dots,j_k);(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)} := \sum_{\mathrm{q}_k} w^{\ell}_{k,i_k,\mathrm{q}_k} b^m_{k,j_k}(\bar{x}^n_{k,q_k}) I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)}\,, \tag{35}$$

and finally for $k = d$:

$$I^{(d)}_{(i_1,\dots,i_d),(j_1,\dots,j_d);()} := \sum_{\mathrm{q}_d} w^{\ell}_{d,i_d,\mathrm{q}_d} b^{m}_{d,j_d}(\bar{x}^{n}_{d,q_d}) I^{(d-1)}_{(i_1,\dots,i_{d-1}),(j_1,\dots,j_{d-1});(\mathrm{q}_d)} = \mathfrak{Q}^{\ell}_{\boldsymbol{i}}(cb^{m}_{\boldsymbol{j}}) \,, \tag{36}$$

where the final expression in (36) is now independent of the quadrature point index but it depends on the pair of test and trial d-tuple $((i_1,\dots,i_d),(j_1,\dots,j_d))$ and is equal to $\mathfrak{Q}^{\ell}_{\boldsymbol{i}}(cb^{m}_{\boldsymbol{j}})$.

Remark 4. For $k = 1,\dots,d$, for each $(k-1)$-tuple pair $(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1})$ the coefficients $I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)}$ are used in a loop of the sum-factorization algorithm, so they should be stored in an efficient data structure for the data retrieval with respect to the loop index $\mathrm{q}_k \in \mathcal{Q}^{n,\ell,m}_{k,i_k,j_k}$. The non-zero entries of $I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)}$ have a non tensor product structure with respect to the point indices $(\mathrm{q}_k,\dots,\mathrm{q}_d)$. Hence, in order to save memory it is better to store these non-zero values in a sparse container. By observing that the indices in $\mathcal{Q}^{n,\ell,m}_{k,i_k,j_k}$ are contiguous, in order to have data locality and improve the cache efficiency, we store the data using a sparse data structure for the data associated to the *inactive summation/integration directions* $k+1,\dots,d$, while the entries along the *active summation/integration direction* k (and associated to the sub-index $(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)$) are stored using a (dense) vector of length equal to the distance between the first and last non-zero value.

The key point to reduce the computational cost is to exploit the fact that the value of $I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)}$ in (35) may be needed to compute multiple values of $I^{(k)}_{(i_1,\dots,i_k),(j_1,\dots,j_k);(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)}$. To exploit this fact, when we are integrating along a direction k we must consider all the pairs of k-tuples $((i_1,\dots,i_k),(j_1,\dots,j_k))$.

Accordingly, for each $k \in \{1,\dots,d\}$ we define the "projection" of the connectivity $K^{n}_{\ell,m}$ along the first k directions:

$$\Pi^{(k)} K^{n}_{\ell,m} := \left\{ \big((i_1,\dots,i_k),(j_1,\dots,j_k)\big) = \big(\tau_k \boldsymbol{i}, \tau_k \boldsymbol{j}\big) \,,\ \forall \big(\boldsymbol{i},\boldsymbol{j}\big) \in K^{n}_{\ell,m} \right\} \tag{37}$$

and then the pairs of k-tuple that must be considered for the efficient computation of (35) are just the elements of $\Pi^{(k)} K^{n}_{\ell,m}$.

The sum-factorization algorithm is summarized by Algorithm 4. The overall algorithm for the matrix formation is depicted by Algorithm 5.

Algorithm 4: `sum_factorization`

Input : $\mathcal{I}^{n}_{\mathcal{Q}}$, $\left\{C^{n}_{\mathbf{q}}\right\}_{\mathbf{q}\in\mathcal{I}^{n}_{\mathcal{Q}}}$, ℓ, $K^{n}_{\ell,m}$, $\left\{\mathcal{W}^{n}_{k}\right\}_{k=1}^{d}$

foreach $(\mathrm{q}_1,\dots,\mathrm{q}_d) \in \mathcal{I}^{n}_{\mathcal{Q}}$ **do**
  $I^{(0)}_{(),();(\mathrm{q}_1,\dots,\mathrm{q}_d)} = C^{n}_{(\mathrm{q}_1,\dots,\mathrm{q}_d)}$
end foreach
for $k = 1,\dots,d$ **do**
  compute $\Pi^{(k)} K^{n}_{\ell,m}$ from (37)
  foreach $(i_k, j_k) \in \{\{1,\dots,N^{\ell}_{\mathcal{B}_k}\} \times \{1,\dots,N^{m}_{\mathcal{B}_j}\} : \mathrm{supp}(b^{\ell}_{k,i_k}) \cap \mathrm{supp}(b^{m}_{k,j_k}) \neq \emptyset\}$ **do**
    compute $\mathcal{Q}^{n,\ell,m}_{k,i_k,j_k}$ from (31);
  end foreach
  foreach $\big((i_1,\dots,i_k),(j_1,\dots,j_k)\big) \in \Pi^{(k)} K^{n}_{\ell,m}$ **do**
    retrieve $\mathcal{W}^{\ell,n}_{k,i_k}$ from $\mathcal{W}^{n}_{k}$ (see (26))
    foreach $(\mathrm{q}_{k+1},\dots,\mathrm{q}_d) \in \{1,\dots,R^{n}_{k+1}\} \times \dots \times \{1,\dots,R^{n}_{d}\}$ **do**
      compute $I^{(k)}_{(i_1,\dots,i_k),(j_1,\dots,j_k);(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)}$ from (35)
    end foreach
  end foreach
end for

Output: $I^{(d)} \equiv \left\{I^{(d)}_{(\boldsymbol{i}),(\boldsymbol{j});()}\right\}_{(\boldsymbol{i},\boldsymbol{j})\in K^{n}_{\ell,m}}$

Algorithm 5: `compute_matrix`

Input : $\mathcal{H}, \{F^n, \mathcal{Q}^n, \mathcal{Q}^n_\Psi, \mathcal{W}^n_1, \dots, \mathcal{W}^n_d\}_{n=0}^L, c$

foreach $n \in \{0, \dots, L\}$ *such that* $F^n \neq \emptyset$ **do**
  compute $\left\{C^n_{\mathbf{q}}\right\}_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}}$ from (32)
  foreach $\ell \in \{0, \dots, n\}$ *such that* $\mathcal{H}^\ell \cap F^n \neq \emptyset$ **do**
    foreach $m \in \{0, \dots, n\}$ **do**
      compute $K^n_{\ell,m}$ from (33)
      $\left\{[M]_{(\ell,i),(m,j)}\right\}_{(i,j)\in K^n_{\ell,m}} =$ `sum_factorization`$(\mathcal{I}^n_{\mathcal{Q}}, \left\{C^n_{\mathbf{q}}\right\}_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}}, \ell, K^n_{\ell,m}, \left\{\mathcal{W}^n_k\right\}_{k=1}^d)$ (Alg. 4)
    end foreach
  end foreach
end foreach

Output: M

3.4. The stiffness matrix

We now briefly discuss how weighted quadrature can be used to form the stiffness matrix, highlighting the main differences with the case of the mass matrix.

Let $\mathfrak{Q}_{\boldsymbol{i}}^{\ell,\beta,\gamma}$ and $\mathfrak{Q}_{\boldsymbol{i}}^{\ell,\gamma}$, for $\beta, \gamma = 1, \dots, d$, denote the hierarchical counterpart of the tensor product rules defined in (7), associated to $b^\ell_{\boldsymbol{i}} \in \mathcal{H}$. Similarly as in the mass case, these rules are defined as a linear combination of tensor product rules on the finest level with whom the basis spline $b^\ell_{\boldsymbol{i}}$ interacts, i.e.

$$\mathfrak{Q}_{\boldsymbol{i}}^{\ell,\beta,\gamma} = \sum_{\boldsymbol{j}\in\mathcal{I}^\nu_{\mathcal{B}}} \alpha^{\ell,\nu}_{\boldsymbol{i},\boldsymbol{j}}\, \bar{\mathfrak{Q}}_{\boldsymbol{j}}^{\nu,\beta,\gamma}, \qquad \mathfrak{Q}_{\boldsymbol{i}}^{\ell,\gamma} = \sum_{\boldsymbol{j}\in\mathcal{I}^\nu_{\mathcal{B}}} \alpha^{\ell,\nu}_{\boldsymbol{i},\boldsymbol{j}}\, \bar{\mathfrak{Q}}_{\boldsymbol{j}}^{\nu,\gamma},$$

where the coefficients $\alpha^{\ell,\nu}_{\boldsymbol{i},\boldsymbol{j}}$ are defined as in (13).

We refer to [11] for the details on the rules $\bar{\mathfrak{Q}}_{\boldsymbol{j}}^{\nu,\beta,\gamma}$ and $\bar{\mathfrak{Q}}_{\boldsymbol{j}}^{\nu,\gamma}$ defined on the tensor product space $\mathcal{B}^n$, for $n = 0, 1, \dots, L$. In particular, the set of weights associated to one such rule is the tensor product of d sets of univariate weights that can be of 4 different types, namely $\{\bar{w}^{n,1,1}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$, $\{\bar{w}^{n,1,0}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$, $\{\bar{w}^{n,0,1}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$ and $\{\bar{w}^{n}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$, for $j_k \in D^n_k$, $k = 1, \dots, d$. The latter set of weights is the same used for the mass matrix, while the former ones are defined by the following univariate exactness conditions

$$\begin{aligned}
\sum_{q_k\in A^{n,n}_{k,j_k}} \bar{w}^{n,1,1}_{k,j_k,q_k} \left(b^n_{k,t_k}\right)'(\bar{x}^n_{k,q_k}) &= \int_0^1 \left(b^n_{k,j_k}\right)'(\xi)\, \left(b^n_{k,t_k}\right)'(\xi)\, d\xi,\\
\sum_{q_k\in A^{n,n}_{k,j_k}} \bar{w}^{n,0,1}_{k,j_k,q_k} \left(b^n_{k,t_k}\right)'(\bar{x}^n_{k,q_k}) &= \int_0^1 b^n_{k,j_k}(\xi)\, \left(b^n_{k,t_k}\right)'(\xi)\, d\xi,\\
\sum_{q_k\in A^{n,n}_{k,j_k}} \bar{w}^{n,1,0}_{k,j_k,q_k} b^n_{k,t_k}(\bar{x}^n_{k,q_k}) &= \int_0^1 \left(b^n_{k,j_k}\right)'(\xi)\, b^n_{k,t_k}(\xi)\, d\xi, \qquad b^n_{k,t_k} \in B^n_{k,j_k},
\end{aligned} \tag{38}$$

where the sets B^n_{k,j_k} and $A^{n,n}_{k,j_k}$ are defined as in (24) and (25).

The pipeline to assemble the stiffness matrix for hierarchical B-splines using WQ is similar to the already presented mass case. The main differences are that in Algorithm 2, the linear systems (38) are solved in addition to (23) to compute all the necessary weights, and that in the innermost loop of Algorithm 5, the contributions of all rules $\mathfrak{Q}_{\boldsymbol{i}}^{\ell,\beta,\gamma}$, $\mathfrak{Q}_{\boldsymbol{i}}^{\ell,\gamma}$ and $\mathfrak{Q}_{\boldsymbol{i}}^{\ell}$ for $\beta, \gamma = 1, \dots, d$ must be computed and summed.

Remark 5. In our implementation, the weights $\{\bar{w}^{n,1,1}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$ and $\{\bar{w}^{n,0,1}_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$ are not actually computed by solving the first and second sets of equations in (38). Note that these equations must be satisfied whenever the integrand

function is the derivative of a B-spline, and these functions span the space of splines with degree and regularity reduced by 1. Therefore, the exactness conditions can be reformulated using the basis of the latter space as integrand functions, leading to equivalent linear systems that is solved to compute the weights.

4. Computational cost

We now want to estimate the total computational cost of the matrix formation, by first focusing on the mass. There are mainly three steps that contribute to this cost: the evaluation of the non-tensor product coefficient c, the computation of the weights, and the computation of the matrix entries via sum-factorization.

The coefficient c has to be evaluated for every active quadrature point. Quadrature points are more dense for elements that are adjacent to the boundary of Ω. However, the total number of active quadrature points is commonly dominated from the interior part. Recalling (14) and $\nu - \ell \leq r - 1$, the number of quadrature points that belong to interior elements is bounded by

$$\sum_{(\ell,\boldsymbol{i})\in\mathcal{I}_{\mathcal{H}}} \#\mathcal{Q}^{(\ell,\boldsymbol{i})} \leq \sum_{(\ell,\boldsymbol{i})\in\mathcal{I}_{\mathcal{H}}} (2^{r-1}(p+1))^d = O(2^{dr}\, p^d N_{\mathcal{H}}). \tag{39}$$

We remark that bound above is not sharp especially for what concerns its dependence on p, since quadrature points in different $\mathcal{Q}^{(\ell,\boldsymbol{i})}$ may coincide.

As for the computation of the weights, we recall that we have to solve a system of the form (23) for every univariate index j_k, for $k = 1, \dots, d$. Since the number of univariate indices is bounded by the number of multi-indices $N_{\mathcal{H}}$, and since each of these linear system has $O(p)$ unknown non-zero weights and $O(p)$ equations, the cost to compute them all using a direct solver is bounded by $O(p^3 N_{\mathcal{H}})$ flops.

If we compare the bound on this cost with the one on the cost to compute the matrix entries (derived below), we see that they have the same order with respect to p for $d = 2$ and that the former has lower order for $d = 3$. Note also that this bound does not depend on the admissibility parameter r.

We finally discuss the computation of the matrix entries. Following the structure of Algorithm 5, we fix $n \in \{0, \dots, L\}$ and $\ell, m \in \{0, \dots, n\}$ and consider the computation of the matrix entries (29) for all $(\boldsymbol{i}, \boldsymbol{j}) \in K^n_{\ell,m}$, as performed by Algorithm 4.

As a preliminary step, we observe that for any fixed direction $k \in \{1, \dots, d\}$ and any fixed index value i_k, the number of indices j_k that must be considered in (30) is clearly bounded by the number of basis functions of level m whose support intersects the support of b^{ℓ}_{k,i_k}. It can be verified that the latter number is bounded by $2p+1$ when $m \leq \ell$, and by $2^{m-\ell}(p+1) + p$ when $m > \ell$. In both cases, this number is bounded by $2^{n-\ell+1}(p+1)$, since $n \geq \max\{m, \ell\}$.

Moreover, again for any fixed direction k and index value i_k, the active quadrature points $\bar{x}^n_{k,q_k}$ are the ones belonging to the support of b^{ℓ}_{k,i_k}; since we have 2 quadrature points on each interior element of level n, or $p+1$ on the elements that touch the boundary, and the support of b^{ℓ}_{k,i_k} contains at most $2^{n-\ell}(p+1)$ elements of level n, we conclude that there are at most $2^{n-\ell+1}(p+1)$ active quadrature points if b^{ℓ}_{k,i_k} does not touch the boundary, or at most $\left(2^{n-\ell+1}+1\right)(p+1)$ quadrature points if b^{ℓ}_{k,i_k} touches the boundary. Again, the cost is typically dominated by the quadrature at the interior, therefore we assume that the number of index values taken by q_k in the kth sum of (30) is roughly $2^{n-\ell+1}(p+1)$.

We are now ready to estimate the cost of computing (30). As a first step, we evaluate the innermost sum (34) for all relevant values of i_1, j_1 and $\mathrm{q}_2, \dots, \mathrm{q}_d$. Of course in the sum we only need to consider the non-zero terms, and we observe that the term corresponding to a fixed q_1 is non-zero only for the $p+1$ values of j_1 such that $b^m_{1,j_1}(\bar{x}^n_{\mathrm{q}_1}) \neq 0$. Note that if we preliminary multiply $w^{\ell}_{1,i_1,\mathrm{q}_1} b^m_{1,j_1}(\bar{x}^n_{\mathrm{q}_1})$ for all such values of q_1 and j_1 (which has a negligible cost), the computation of the sum (34) requires 2 flops for each of its non-zero terms.

Since each index $\mathrm{q}_1, \dots, \mathrm{q}_d$, can take up to $2^{n-\ell+1}(p+1)$ values, and since the number of values taken by $i_1 = \tau_1 \boldsymbol{i}$ is bounded by the number of multi-indices $\boldsymbol{i}$ belonging to $F^n \cap \mathcal{H}^{\ell}$, the cost of the first step is bounded by

$$2\,(p+1)^{d+1}\; 2^{d(n-\ell+1)} N_{n,\ell} \text{ flops}, \tag{40}$$

where

$$N_{n,\ell} := |F^n \cap \mathcal{H}^{\ell}|.$$

For $k = 2, \dots, d-1$, the kth step of the sum-factorization requires the computation of (35) for all values of $i_1, \dots, i_k$, $j_1, \dots, j_k$ and $\mathrm{q}_{k+1}, \dots, \mathrm{q}_d$, where the inner sum $I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)}$ has already been computed for all the relevant index values.

Since $(i_1, \dots, i_k) = \tau_k \boldsymbol{i}$, the total number of k-tuples $(i_1, \dots, i_k)$ that have to be considered is again bounded by the number of multi-indices $N_{n,\ell}$.

Moreover, again we observe that for each value of q_k there are only $p+1$ values of j_k that contribute to the sum, and since the number of values taken by each index $\mathrm{q}_k, \dots, \mathrm{q}_d$ and $j_1, \dots, j_{k-1}$ is bounded by $2^{n-\ell+1}(p+1)$, the cost of this step is again bounded by (40). With similar arguments, it can be shown that this is true also for the dth step of the sum-factorization (35).

We conclude that the cost of the whole sum-factorization step is bounded by

$$2d\,(p+1)^{d+1}\;2^{d(n-\ell+1)}N_{n,\ell}\ \text{flops}.$$

We sum the above expression for all values of n, ℓ and m, and observe that for a fixed level ℓ the number of levels m that interact with it is at most $2r-1$. Thus, a bound on the total cost for the matrix entries computation is given by

$$2d(2r-1)\,(p+1)^d \sum_n \sum_{\ell \le n} 2^{d(n-\ell+1)} N_{n,\ell}\ \text{flops}. \tag{41}$$

We can derive a more explicit bound on the cost of the matrix entries computation if take a further step and observe that $n - \ell + 1 \le r$ and that

$$\sum_n \sum_{\ell \le n} N_{n,\ell} = N_{\mathcal{H}}.$$

Hence the total cost for the matrix entries computation (41) is bounded by

$$2d(2r-1)2^{dr}\,(p+1)^{d+1}\,N_{\mathcal{H}} = O\left(dr2^{dr}\,p^{d+1}N_{\mathcal{H}}\right)\ \text{flops}. \tag{42}$$

We observe that, similarly as in the bound on the active quadrature points (39), the latter expression grows exponentially with respect to the admissibility parameter r, and this effect worsen with the increasing of the dimension d. This might seem unsatisfactory, but we emphasize that (42) is easily a rather pessimistic bound. Indeed, a careful analysis of the derivation of (42) reveals that we are essentially assuming that every hierarchical B-spline basis function $b^{\ell}_{\boldsymbol{i}}$, with $(\ell, \boldsymbol{i}) \in \mathcal{I}_{\mathcal{H}}$, interacts with all the admissible levels. In many practical cases, however, refinement is performed only in specific regions of the domain, e.g., in the neighborhood of low dimensional manifolds, and as a result the number of basis functions that interact with all the admissible levels is limited.

The computational cost for the stiffness matrix can be analyzed in a similar way, since the number of quadrature points is the same. The main difference is that each integral appearing in (6) has to be approximated independently, and this clearly increases the cost. For example, if only second-order terms appear in the operator (as is the case for the scalar Poisson stiffness matrix), the formation cost is d^2 times higher than for the mass matrix.

5. Numerical tests

The numerical tests comprise of the L^2-projection operator (both in 2D and 3D) $a(u,v) := \int_\Omega uv \;d\Omega$, and the linear elasticity operator (in 3D) $a(\boldsymbol{u},\boldsymbol{v}) := 2\mu \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \;d\Omega + \lambda \int_\Omega \mathrm{div}(\boldsymbol{u}) \cdot \mathrm{div}(\boldsymbol{v}) \;d\Omega$, where $\boldsymbol{\varepsilon}_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$.

For the 2D cases the physical domain $\Omega \in \mathbb{R}^2$ is defined as the image of the two-dimensional parametric domain $\hat{\Omega} = [1,2] \times [\frac{\pi}{4}, \frac{3\pi}{4}]$ through the (polar) map

$$F(\rho,\theta) = \begin{pmatrix} \rho\cos\theta \\ \rho\sin\theta \end{pmatrix}, \tag{43}$$

while for the 3D cases the physical domain $\Omega \subset \mathbb{R}^3$ is defined as the image of the three-dimensional parametric domain $\hat{\Omega} = [1,2] \times [\frac{\pi}{4}, \frac{3\pi}{4}] \times [0, \frac{\pi}{2}]$ through the (polar) map

$$F(\rho,\theta,\phi) = \begin{pmatrix} \rho\cos\theta \\ \rho\sin\theta\cos\phi \\ \rho\sin\theta\sin\phi \end{pmatrix}. \tag{44}$$

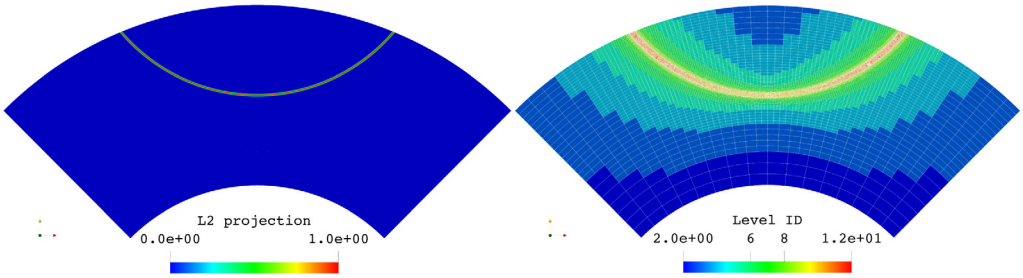

Fig. 3. L^2-projection (left) and element levels (right) after 51 adaptive refinements for the 2D case with degree $p = 2$ and admissibility $r = 2$. For this configuration the space contains 867947 degrees of freedom, and the L^2-error between the function (45) and its L^2-projection is $\approx 3.4e-7$.

For both L^2-projection and linear elasticity problems we set the boundary conditions and the right-hand side vector of the linear system that assures a known unique exact solution of the problem (the details are given below) and then, for a given value of the admissibility parameter r, a nested sequence of hierarchical B-spline spaces of degree p is constructed [31]. The adaptive mesh refinement is steered by the "error estimator", which is simply the L^2-error between the computed solution and the known solution and by using the Dörfler marking strategy [33] with parameter $\theta_* = 0.2$.

For each refinement step, we perform a simulation using the standard element-based Gaussian quadrature (using $p + 1$ quadrature point along each direction of the element) to build the bilinear operator (and the vector of the linear system) and then, using the same sequence of hierarchical spaces we compute the bilinear operator using the proposed hierarchical WQ algorithm.

Remark 6. All the numerical tests were performed using the IGATOOLS library [34], on a single core of an Intel Xeon Gold 6242R processor running at 3.1 GHz. In order to alleviate the random fluctuations in the elapsed CPU time, all plots involving CPU time refer to the average CPU time of multiple (5 for the 2D case and 3 for the 3D case) runs of the same simulation.

Remark 7. In all plots the lowest monitored CPU time is set to 10^{-1} seconds to reduce the effect of random time fluctuations, due to the CPU scheduling.

5.1. L^2-Projection

These numerical tests comprise of the L^2-projection of the function $f : \Omega \to \mathbb{R}$,

$$f(\mathbf{x}) = \exp\left[-\left(\frac{\|\mathbf{x} - \mathbf{x}_0\| - 1}{\beta}\right)^2\right], \tag{45}$$

where the physical domain $\Omega \in \mathbb{R}^d$ is specified above ((43)–(44)), the parameter $\beta \in \mathbb{R}$ and the point $\mathbf{x}_0 \in \mathbb{R}^d$ are:

- $\beta = 5 \cdot 10^{-3}$ and $\mathbf{x}_0 = (0, \frac{5}{2})$ for $d = 2$ (see Fig. 3);
- $\beta = 0.1$ and $\mathbf{x}_0 = (0, \frac{5}{2}, 0)$ for $d = 3$ (see Fig. 8).

5.1.1. L^2-Projection: 2D case

For this case we performed simulations using the admissibility parameters $r = 2, 3$ and for each value of r we used the degrees $p = 2, \ldots, 6$.

Fig. 4 shows the total time (preprocessing + matrix computation) with respect to $N_{\mathcal{H}}$ for $r = 2$. The plots indicate that the total time for the WQ approach seems to be nearly independent of the degree p, while for the element-based Gaussian approach we note that the cost increases with p (as expected). Moreover, also the most favorable case for the element-based Gaussian approach (i.e., $p = 2$) costs more of any of the WQ cases we have tested. As a result, we can conclude that if one wants achieve a very low error level ($< 10^{-7}$), the best strategy in terms of CPU time needed to build the matrix is to use WQ with high degree (see Fig. 5).

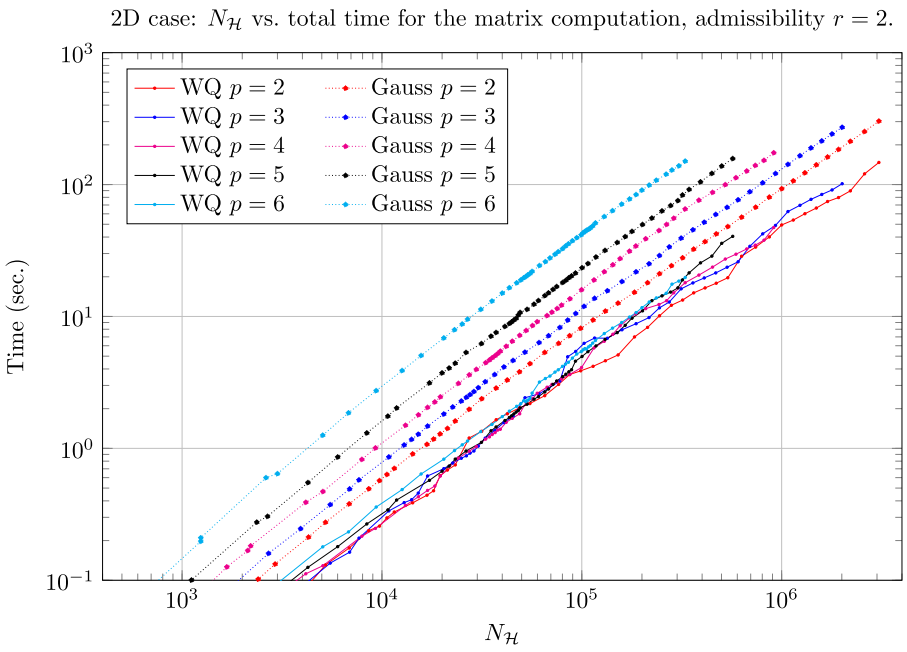

Fig. 4. L^2-projection: $N_{\mathcal{H}}$ vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 2$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

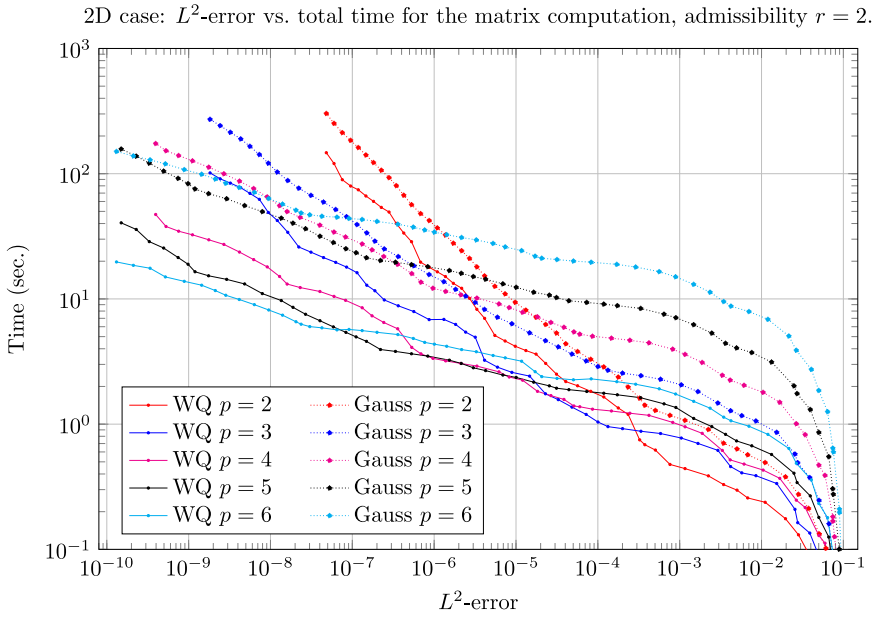

Fig. 5. L^2-projection: L^2-error vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 2$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

Regarding the CPU cost of the WQ approach, in Fig. 6 are shown (for the degrees $p = 2, \ldots, 5$) the preprocessing cost (Algorithm 3) and the matrix computation cost (Algorithm 5), that is split in the time needed to evaluate the coefficients in (32) (for $n = 0, \ldots, L$) and the rest of the algorithm (i.e., the computation of the connectivities $K^n_{\ell,m}$ from (33) and the sum-factorization). From the plots in Fig. 6 we observe that the asymptotic behavior of the costs is the same for all different degrees, resulting in the dominant cost being the formation of the matrix whereas the

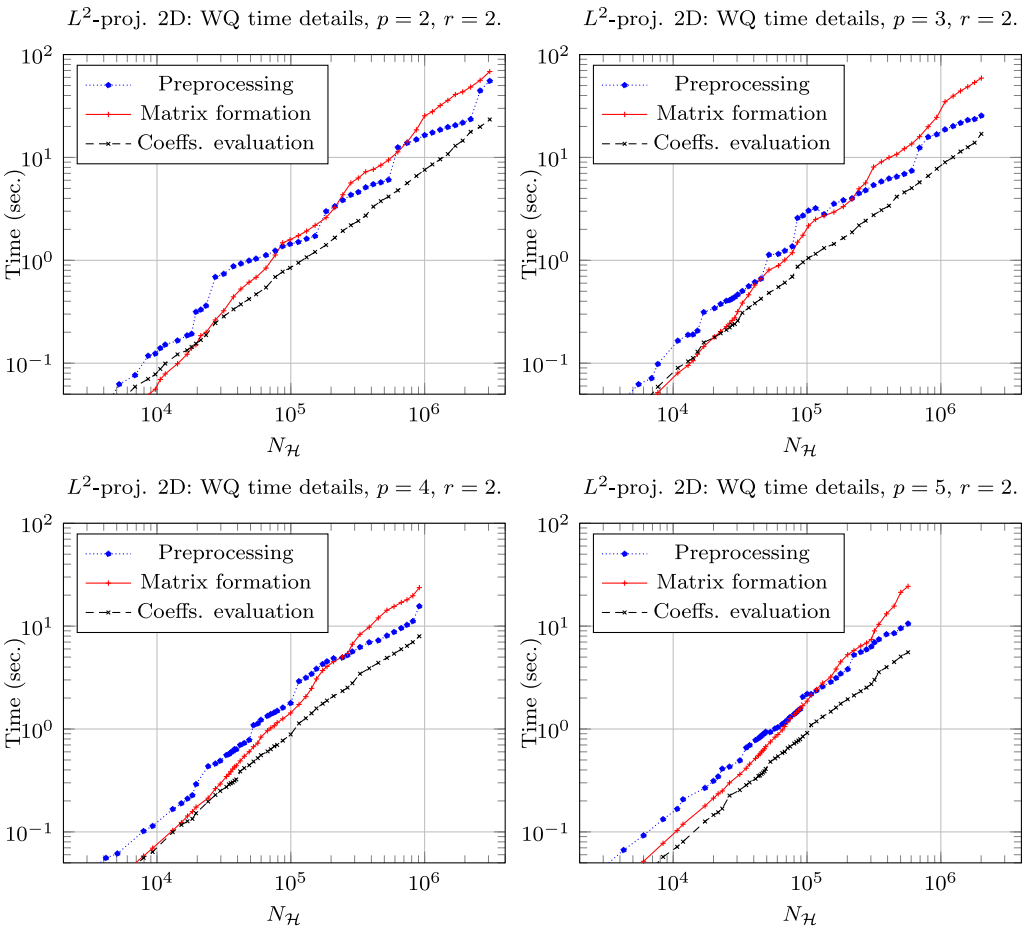

Fig. 6. $N_{\mathcal{H}}$ vs. the CPU time needed to run the WQ algorithm: preprocessing (Algorithm 3) and matrix computation (Algorithm 5) for the L^2-projection in 2D and admissibility parameter $r = 2$. The cost for the matrix computation is split in the CPU cost for coefficient evaluations ((32) for $n = 0, \dots, L$) and the CPU cost for executing the rest of the Algorithm 5.

cost for the preprocessing is smaller but not negligible (at least for the tested cases). It is worthy to note that for low number of degrees of freedom, the main cost is due to the preprocessing.

Regarding the case with admissibility parameter $r = 3$, Fig. 7 shows the total time (preprocessing + matrix computation) with respect to $N_{\mathcal{H}}$. The plots indicate that both approaches (element-based Gaussian quadrature and WQ) have a higher cost (for a given number of $N_{\mathcal{H}}$) for all tested degrees with respect to the case with $r = 2$ (Fig. 4), but the WQ approach seems to be nearly independent of the degree and for the tested degrees $p = 2, \dots, 6$ it is less expensive than the element-based Gaussian approach of degree ≥ 3.

5.1.2. L^2-Projection: 3D case

For this case we performed simulations using the admissibility parameter $r = 2$ and the degrees $p = 2, \dots, 6$.

Fig. 9 shows the total time (preprocessing + matrix computation) with respect to $N_{\mathcal{H}}$, while Fig. 10 shows the total time with respect to the L^2-error. From the plots in these figures, we can observe that the WQ approach outperforms the element-based Gaussian approach. In fact, considering the CPU time with respect to $N_{\mathcal{H}}$ (Fig. 9), the total time (preprocessing + matrix computation) for the WQ approach seems to be mildly dependent from the degree p, while for the element-based Gaussian approach we note that the cost increases with p, by a factor higher than the 2D case (as expected). Moreover, also the most favorable case for the element-based Gaussian approach (i.e. $p = 2$) costs more of any of the WQ cases we have tested. As result, we can again infer that the best strategy in terms of error/CPU time ratio is WQ with high degree (see Fig. 10).

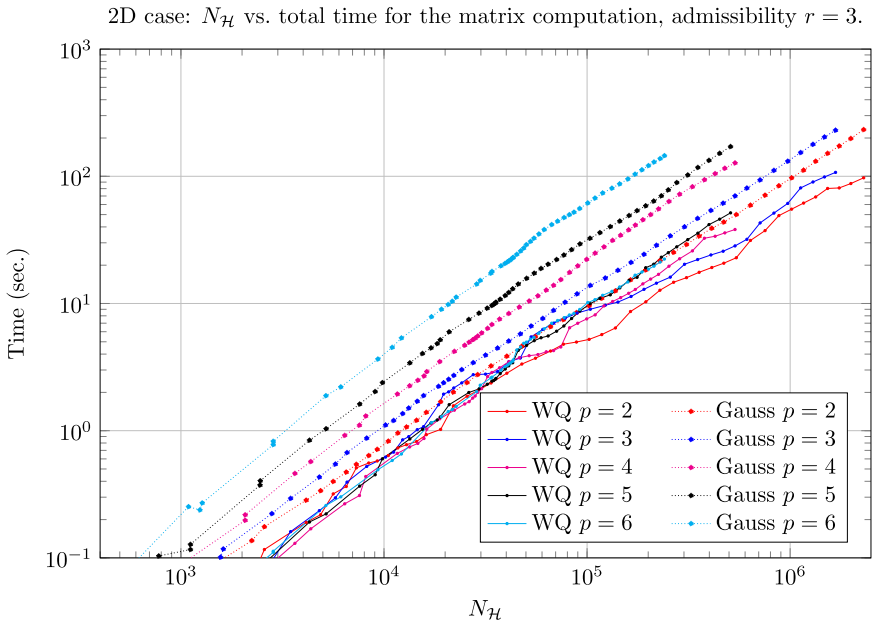

Fig. 7. $N_{\mathcal{H}}$ vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 3$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

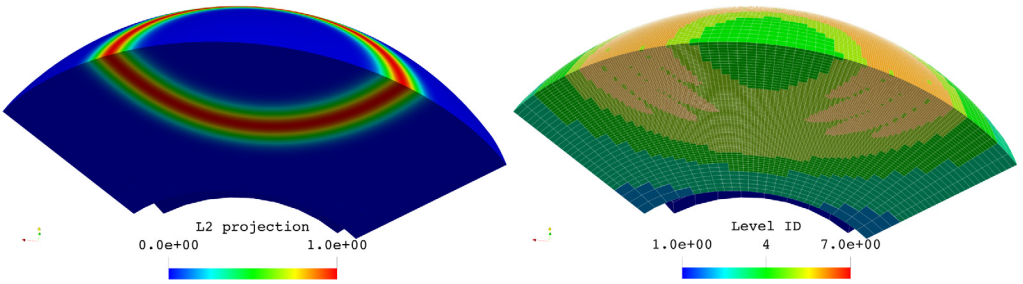

Fig. 8. L^2-projection (left) and element levels (right) after 27 adaptive refinements for the 3D case with degree $p = 2$ and admissibility $r = 2$. For this configuration the space contains 754614 degrees of freedom, and the L^2-error between the function (45) and its L^2-projection is $\approx 8.5e-6$.

Regarding the CPU cost of the WQ approach, in Fig. 11 are shown (for the degrees $p = 2, \dots, 5$) the preprocessing cost (Algorithm 3) and the matrix computation cost (Algorithm 5), that is split in the time needed to evaluate the coefficients in (32) (for $n = 0, \dots, L$) and the rest of the algorithm (i.e., the computation of the connectivities $K^n_{\ell,m}$ from (33) and the sum-factorization). In this case we observe that for degree $p = 2$ the dominant cost is due to the evaluation of the coefficients, whereas for higher degrees this cost is less important (moreover, the relative importance decreases raising the degree, and increases raising the number of degrees of freedom). Regarding the other two major costs, the same pattern observed for the 2D case seems to emerge: when the number of degrees of freedom is "small" (depending on the degree) the bigger cost is due to the preprocessing, otherwise the dominating cost is due to the matrix formation phase.

5.2. Linear elasticity in 3D

These numerical tests comprise of the solution of the following linear elasticity problem:

Find $\boldsymbol{u} \in \left(H_0^1(\Omega)\right)^3$ s.t. $\forall \boldsymbol{v} \in \left(H_0^1(\Omega)\right)^3$ holds

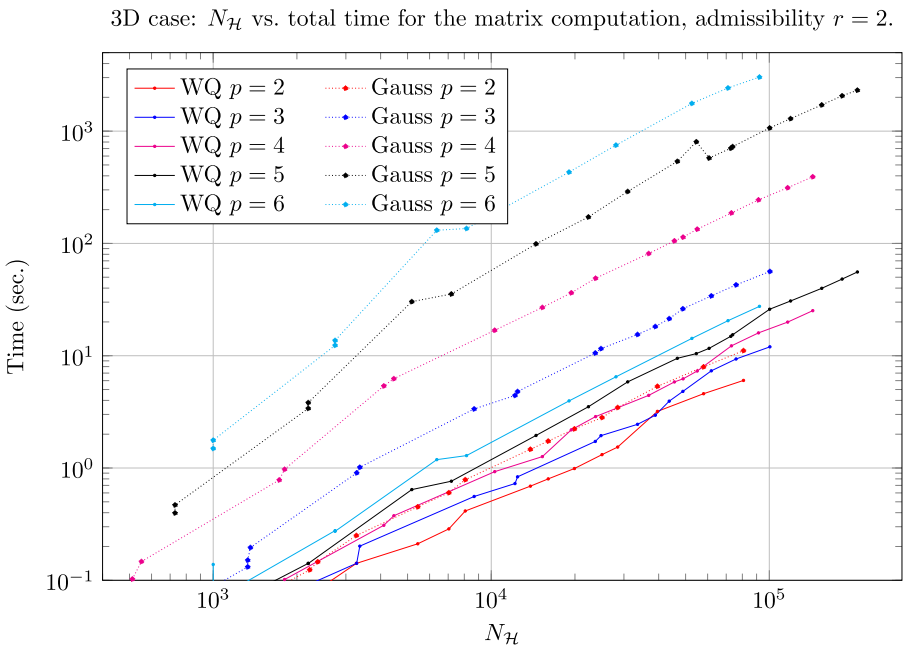

Fig. 9. L^2-projection: $N_{\mathcal{H}}$ vs. total time for the matrix computation for the 3D case and admissibility parameter $r = 2$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

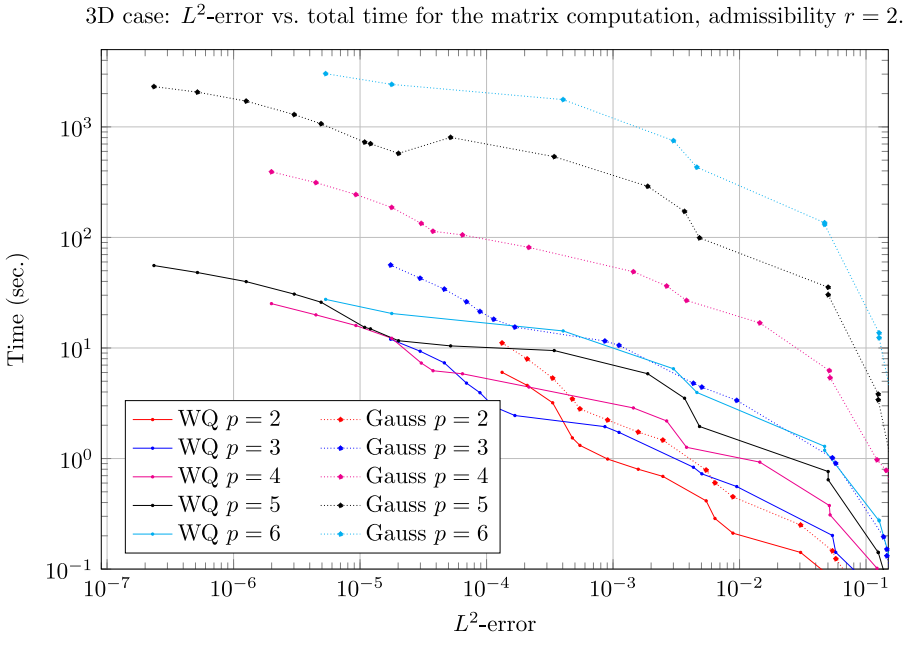

Fig. 10. L^2-projection: L^2-error vs. total time for the matrix computation for the 3D case and admissibility parameter $r = 2$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

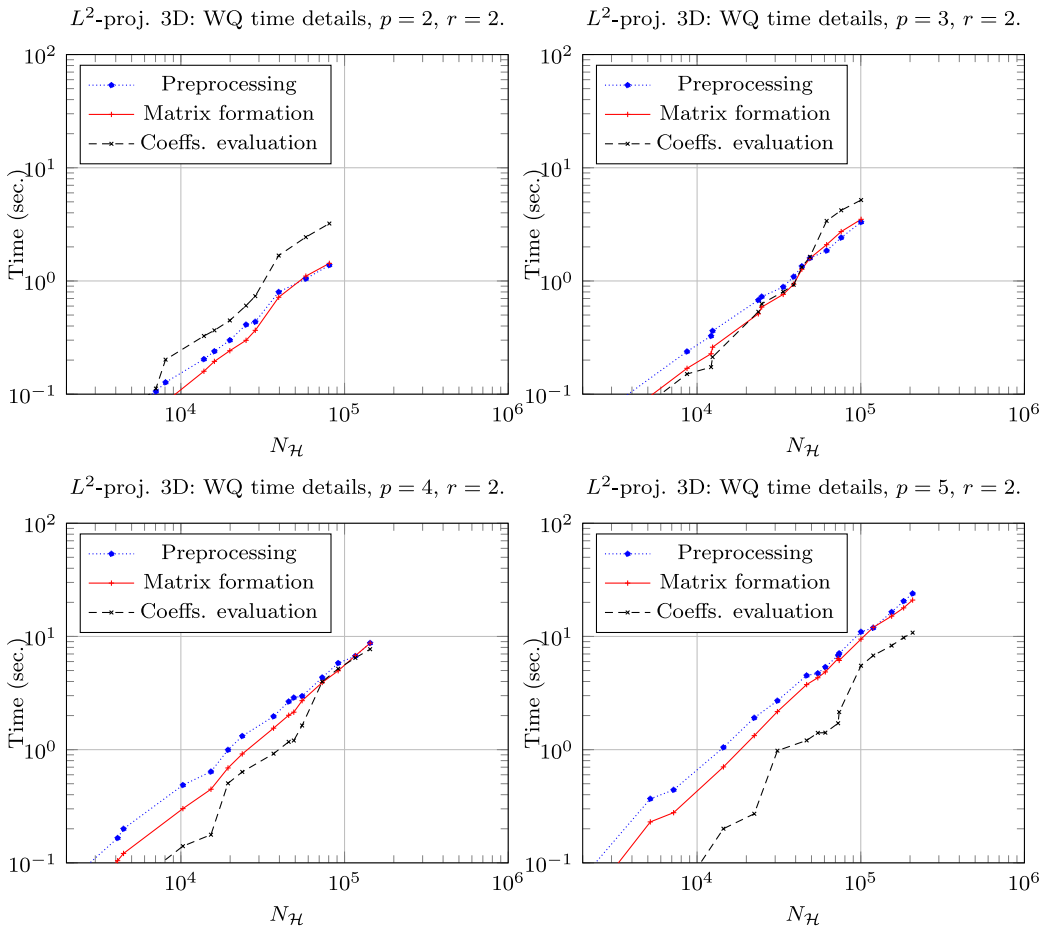

Fig. 11. $N_{\mathcal{H}}$ vs. the CPU time needed to run the WQ algorithm: preprocessing (Algorithm 3) and matrix computation (Algorithm 5) for the L^2-projection in 3D and admissibility parameter $r = 2$. The cost for the matrix computation is split in the CPU cost for coefficient evaluations ((32), for $n = 0, \dots, L$) and the CPU cost for executing the rest of the Algorithm 5.

$$\mu \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v})\, d\Omega + \lambda \int_\Omega \operatorname{div}(\boldsymbol{u}) \cdot \operatorname{div}(\boldsymbol{v})\, d\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v}\, d\Omega \quad \text{on } \Omega \tag{46}$$

where $\boldsymbol{\varepsilon}_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$ and the Lamé parameters (λ, μ) correspond to the elasticity modulus $E = 1$ and Poisson's ratio $\nu = 0.3$. The function $\boldsymbol{f} : \Omega \to \mathbb{R}^3$ is chosen to satisfy the strong form of (46) applied to the manufactured solution $\boldsymbol{u} : \Omega \to \mathbb{R}^3$

$$\boldsymbol{u}(\mathbf{x}) = \exp\left[-\left(\frac{\|\mathbf{x} - \mathbf{x}_0\| - 1}{\beta}\right)^2\right](\rho^2 - 1)(\rho^2 - 4)\left(\frac{\pi}{4} - \theta\right)\left(\frac{3\pi}{4} - \theta\right)\phi\left(\frac{\pi}{2} - \phi\right)(\mathbf{x} - \mathbf{x}_0)\,, \tag{47}$$

where the parameters (ρ, θ, ϕ) are the parametric coordinates of the (physical) point $\mathbf{x}$ when mapped by the function (44).

For this case we performed simulations using the admissibility parameter $r = 2$ and the degrees $p = 2, \dots, 6$.

Fig. 12 shows the total time (preprocessing + matrix computation) with respect to $N_{\mathcal{H}}$. From the plots in this figure we can observe that the WQ approach has a milder dependence on p than the element-based Gaussian approach, and (for a fixed degree p) its total cost is smaller than the cost of the element-based Gaussian approach, except for the degree $p = 2$ where the two costs are comparable. In particular, it is worthy to note that for degree $p = 6$, when $N_{\mathcal{H}} \approx 8.0 \cdot 10^4$, the time for assembling the stiffness matrix for WQ is ≈ 100 times smaller than the time needed by the standard element-based Gaussian approach.

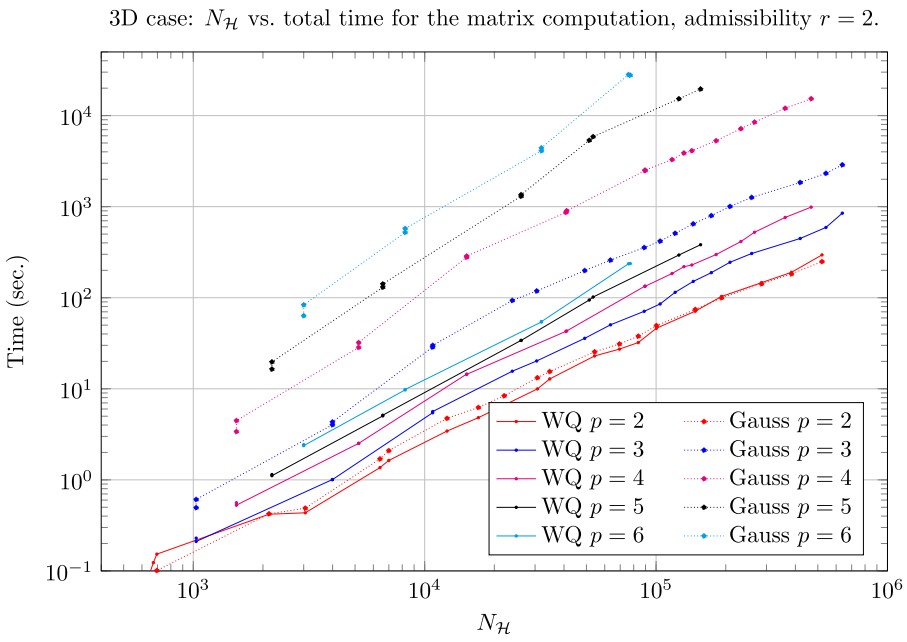

Fig. 12. Linear elasticity: $N_{\mathcal{H}}$ vs. total time for the matrix computation in the 3D case and admissibility parameter $r = 2$. For WQ the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

6. Closure

A fast matrix formation technique for adaptive isogeometric Galerkin methods with multivariate hierarchical B-splines was presented by focusing on the efficient design of weighted quadrature rules. As standard choice for adaptivity with hierarchical B-spline constructions, we perform local mesh refinement by successively introducing dyadically refined knot sequences as background machinery for the construction of the spline hierarchy. The theoretical estimates of the computational cost suitably exploit the limited number of basis functions, which are non-zero on any element of an admissible hierarchical mesh. A selection of numerical examples confirm that the results obtained with the hierarchical weighted approach compare favorably with respect to standard Gaussian quadrature rules, specially in the three-dimensional case. Interesting topics for future research include for example a combination of the proposed algorithm with matrix-free methods [35] as well as the extension to the case of truncated hierarchical B-splines [3,4], characterized by (possibly) reduced supports with respect to the ones of standard hierarchical B-splines, and an application to different PDE problems of applicative interest.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

C. Giannelli, M. Martinelli, G. Sangalli and M. Tani are members of the INdAM Research group GNCS. The INdAM support through GNCS and Finanziamenti Premiali SUNRISE is gratefully acknowledged. Moreover, M. Tani was partially supported by the INdAM-GNCS 2022 project “Metodi numerici efficienti e innovativi per la risoluzione di PDE”.

This work was partially supported by the ERC Project CHANGE, which has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation programme (grant agreement No 694515).

References

[1] A. Buffa, G. Gantner, C. Giannelli, D. Praetorius, R. Vázquez, Mathematical foundations of adaptive isogeometric analysis, Arch. Comput. Methods. Engrg. (2022) in press, arXiv:2107.02023.

[2] 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 (2011) 3554–3567.

[3] C. Giannelli, B. Jüttler, H. Speleers, THB-splines: The truncated basis for hierarchical splines, Comput. Aided Geom. Des. 29 (2012) 485–498.

[4] C. Giannelli, B. Jüttler, S. 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.

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

[6] A. Buffa, C. Giannelli, Adaptive isogeometric methods with hierarchical splines: Optimality and convergence rates, Math. Models Methods Appl. Sci. 27 (2017) 2781–2802.

[7] G. Gantner, D. Haberlik, D. Praetorius, Adaptive IGAFEM with optimal convergence rates: Hierarchical B-splines, Math. Models Methods Appl. Sci. 27 (2017) 2631–2674.

[8] P. Hennig, M. Ambati, L. De Lorenzis, M. Kästner, Projection and transfer operators in adaptive isogeometric analysis with hierarchical B-splines, Comput. Methods Appl. Mech. Engrg. 334 (2018) 313–336.

[9] M. Carraturo, C. Giannelli, A. Reali, R. Vázquez, Suitably graded THB-spline refinement and coarsening: towards an adaptive isogeometric analysis of additive manufacturing processes, Comput. Methods Appl. Mech. Engrg. 348 (2019) 660–679.

[10] 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.

[11] F. Calabrò, G. Sangalli, M. Tani, Fast formation of isogeometric Galerkin matrices by weighted quadrature, Comput. Methods Appl. Mech. Engrg. 316 (2017) 606–622.

[12] M. Pan, B. Jüttler, A. Giust, Fast formation of isogeometric Galerkin matrices via integration by interpolation and look-up, Comput. Methods Appl. Mech. Engrg. 366 (2020) 113005.

[13] M. Pan, B. Jüttler, A. Mantzaflaris, Efficient matrix assembly in isogeometric analysis with hierarchical B-splines, J. Comput. Appl. Math. 390 (2021) 113278.

[14] T. Hirschler, P. Antolin, A. Buffa, Fast and multiscale formation of isogeometric matrices of microstructured geometric models, Comput. Mech. 69 (2022) 439–466.

[15] P. Antolin, A. Buffa, F. Calabro, M. Martinelli, G. Sangalli, Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization, Comput. Methods Appl. Mech. Engrg. 285 (2015) 817–828.

[16] A. Bressan, S. Takacs, Sum factorization techniques in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 352 (2019) 437–460.

[17] D. Drzisga, B. Keith, B. Wohlmuth, The surrogate matrix methodology: Accelerating isogeometric analysis of waves, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113322.

[18] G. Moutsanidis, W. Li, Y. Bazilevs, Reduced quadrature for FEM, IGA and meshfree methods, Comput. Methods Appl. Mech. Engrg. 373 (2021) 113521.

[19] F. Fahrendorf, L. De Lorenzis, H. Gomez, Reduced integration at superconvergent points in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 328 (2018) 390–410.

[20] Z. Zou, T. Hughes, M. Scott, R. Sauer, E. Savitha, Galerkin formulations of isogeometric shell analysis: Alleviating locking with Greville quadratures and higher-order elements, Comput. Methods Appl. Mech. Engrg. 380 (2021) 113757.

[21] A. Mantzaflaris, B. Jüttler, B.N. Khoromskij, U. Langer, Low rank tensor methods in Galerkin-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 316 (2017) 1062–1085.

[22] A. Karatarakis, P. Karakitsios, M. Papadrakakis, GPU accelerated computation of the isogeometric analysis stiffness matrix, Comput. Methods Appl. Mech. Engrg. 269 (2014) 334–355.

[23] T.J. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 199 (5–8) (2010) 301–313.

[24] F. Auricchio, F. Calabro, T.J. Hughes, A. Reali, G. Sangalli, A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 249 (2012) 15–27.

[25] M. Bartoň, V.M. Calo, Gauss–Galerkin quadrature rules for quadratic and cubic spline spaces and their application to isogeometric analysis, Comput.-Aided Des. 82 (2017) 57–67.

[26] M. Bartoň, V.M. Calo, Optimal quadrature rules for odd-degree spline spaces and their application to tensor-product-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 305 (2016) 217–240.

[27] R.R. Hiemstra, F. Calabro, D. Schillinger, T.J. Hughes, Optimal and reduced quadrature rules for tensor product and hierarchically refined splines in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 316 (2017) 966–1004.

[28] D. Schillinger, S.J. Hossain, T.J. Hughes, Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 277 (2014) 1–45.

[29] C. Adam, T.J. Hughes, S. Bouabdallah, M. Zarroug, H. Maitournam, Selective and reduced numerical integrations for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 284 (2015) 732–761.

[30] R.R. Hiemstra, G. Sangalli, M. Tani, F. Calabro, T.J. Hughes, Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity, Comput. Methods Appl. Mech. Engrg. 355 (2017) 234–260.

[31] C. Bracco, C. Giannelli, R. Vázquez, Refinement algorithms for adaptive isogeometric methods with hierarchical splines, Axioms 7 (3) (2018) 43.
[32] B. Marussig, Fast formation and assembly of isogeometric Galerkin matrices for trimmed patches, in: C. Manni, H. Speleers (Eds.), Geometric Challenges in Isogeometric Analysis, Springer INdAM Series, Vol. 49, 2022.
[33] W. Dörfler, A convergent algorithm for Poisson's equation, SIAM J. Numer. Anal. 33 (1996) 1106–1124.
[34] M.S. Pauletti, M. Martinelli, N. Cavallini, P. Antolin Sanchez, Igatools: an isogeometric analysis library, SIAM J. Sci. Comput. 37 (4) (2015) 465–496.
[35] G. Sangalli, M. Tani, Matrix-free weighted quadrature for a computationally efficient isogeometric k-method, Comput. Methods Appl. Mech. Engrg. 338 (2018) 117–133.