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, 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*

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 constructions [2, 3, 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, 6, 7] to engineering applications, see e.g., [8, 9, 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

Email addresses: carlotta.giannelli@unifi.it (Carlotta Giannelli), tadej.kanduc@fmf.uni-lj.si (Tadej Kanduč), martinelli@imati.cnr.it (Massimiliano Martinelli), giancarlo.sangalli@unipv.it (Giancarlo Sangalli), mattia.tani@imati.cnr.it (Mattia Tani)

September 28, 2021

interpolation and look-up [12, 13], multiscale quadrature [14], sum factorization [15, 16], the surrogate matrix method [17], reduced integration at superconvergent points [18] and, beyond quadrature, the use of low-rank approximation [19] or GPUs [20].

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 Gauss quadrature, generalized Gauss quadrature [21, 22, 23, 24] or reduced quadrature [25, 26, 27]. Therefore, WQ requires less quadrature points, which mildly depend on the spline degree.

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. To illustrate the performance of the method and confirm the theoretical estimates, a selection of 2D and 3D numerical tests is provided. For the sake of simplicity, we discuss the case of the mass matrix and L^2-projection. However, dealing with other matrices and with PDE problems is conceptually the same, see [11].

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 mass matrix formation for hierarchical B-splines is then presented in Section 3, while its 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 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 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,

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

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

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

with respect to the index set

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

We denote the rectilinear mesh grid of level ℓ by $\mathcal{M}^\ell$. The dimension of V^ℓ is then simply given by

$$N_{\mathcal{B}}^\ell := \dim\left(\mathcal{B}^\ell\right) = \prod_{k=1}^{d} \dim\left(\mathcal{B}_k^\ell\right) = \prod_{k=1}^{d} N_{\mathcal{B}_k}^\ell \ .$$

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 $m > \ell$,

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

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

$$\mathcal{H}(\mathcal{M}) := \left\{ b_{\boldsymbol{i}}^\ell \in \mathcal{B}^\ell \ : \ \boldsymbol{i} \in \mathcal{I}_{\mathcal{H}}^\ell, \, \ell = 0, \ldots, L \right\},$$

where

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

The cardinality is denoted by $N_{\mathcal{H}}$. Each basis function $b_{\boldsymbol{i}}^\ell \in \mathcal{H}$ is uniquely identified by its level ℓ and by the multi-index $\boldsymbol{i} \in \mathcal{I}_{\mathcal{H}}^\ell$. Hence we can define the set of basis identifiers

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

Hierarchical B-splines are non-negative, linear independent, and allow localized mesh refinement by suitably selecting basis functions with a varying level of resolution.

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 [5, 1] for more details concerning admissible meshes and their properties, and to [28] 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 degree and degrees-of-freedom number, 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 a 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

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

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 k-th direction and R_k^ℓ is the number of quadrature points along that direction. In the case of splines of

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 $p+1$ uniformly distributed quadrature points can be selected at each edge of the interval, see [11]. We also introduce the tensor product set of multi-indices, associated to $\mathcal{Q}^\ell$ as

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

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 $\boldsymbol{i}$. This is formalized by setting to 0 the weights $\{\bar{w}_{\boldsymbol{i},\mathbf{q}}^\ell\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^\ell}$ corresponding to points that are outside the support of $b_{\boldsymbol{i}}^\ell$. The nonzero 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.

3. WQ and mass matrix formation for hierarchical B-splines

3.1. Definition of WQ for hierarchical B-splines

A weighted quadrature rule, associated to an active basis function $b_{\boldsymbol{i}}^\ell \in \mathcal{H}$, is denoted by $\mathfrak{Q}_{\boldsymbol{i}}^\ell$. 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

$$\mathcal{Q}^{(\ell,\boldsymbol{i})} \coloneqq \left\{\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,i)}} \tag{6}$$

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

$$\mathfrak{Q}_{\boldsymbol{i}}^\ell(v) := \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,i)}} w_{\boldsymbol{i},\mathbf{q}}^\ell v\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right) \approx \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) v(\boldsymbol{x}) d\boldsymbol{x}. \tag{7}$$

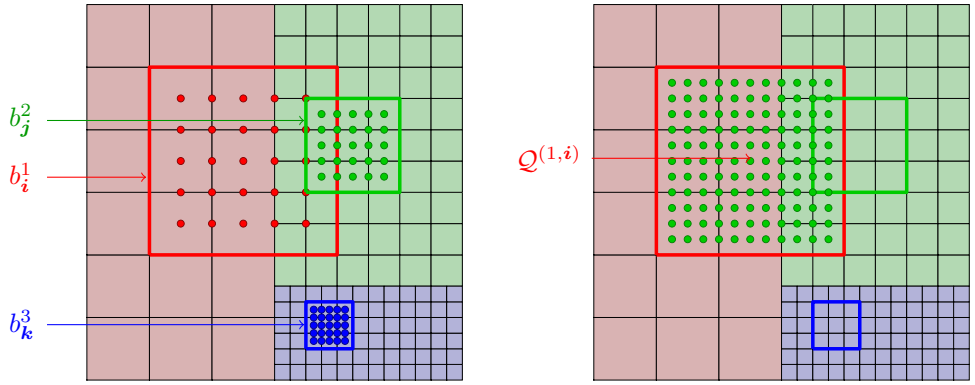
A peculiar feature of this structure is that both the set of quadrature points and the set of quadrature weights depend on the considered test function $b_{\boldsymbol{i}}^\ell$. However, as we will see in the following, $\mathcal{Q}^{(\ell,\boldsymbol{i})}$ can be conveniently selected as a subset of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

$$\mathfrak{Q}_{\boldsymbol{i}}^\ell(b_{\boldsymbol{j}}^m) = \sum_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{(\ell,i)}} w_{\boldsymbol{i},\mathbf{q}}^\ell \, b_{\boldsymbol{j}}^m\left(\boldsymbol{x}_{\boldsymbol{i},\mathbf{q}}^\ell\right) = \int_{[0,1]^d} b_{\boldsymbol{i}}^\ell(\boldsymbol{x}) b_{\boldsymbol{j}}^m(\boldsymbol{x}) d\boldsymbol{x} \qquad \forall \, (m,\boldsymbol{j}) \in \mathcal{I}_{\mathcal{H}}. \tag{8}$$

For a given pair $(\ell,\boldsymbol{i}) \in \mathcal{I}_{\mathcal{H}}$ we define $\nu(\ell,\boldsymbol{i})$ as the finest level of a hierarchical basis function such that its support has a nonempty intersection with the support of $b_{\boldsymbol{i}}^\ell$, i.e.,

$$\nu(\ell,\boldsymbol{i}) \coloneqq \max\left\{m \ : \ \mathrm{supp}(b_{\boldsymbol{i}}^\ell) \cap \mathrm{supp}(b_{\boldsymbol{j}}^m) \neq \emptyset, \ \forall b_{\boldsymbol{j}}^m \in \mathcal{H}\right\}. \tag{9}$$



(a) Hierarchical mesh and three basis functions from $\mathcal{B}^\ell$

(b) Hierarchical quadrature points for $b^1_{\boldsymbol{i}} \in \mathcal{H}$

Figure 1: In (a), the supports and quadrature points for three basis functions from spaces $\mathcal{B}^1$, $\mathcal{B}^2$ and $\mathcal{B}^3$ are depicted on a hierarchical mesh, color coded in red, green and blue, respectively. In (b), quadrature points for $b^1_{\boldsymbol{i}} \in \mathcal{H}$ are shown.

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 $b^m_{\boldsymbol{j}}$ that interacts with $b^\ell_{\boldsymbol{i}}$ (including itself) can be written as a linear combination of basis functions of level $\nu(\ell, \boldsymbol{i})$, that is

$$b^m_{\boldsymbol{j}} = \sum_{\boldsymbol{t} \in \mathcal{I}^\nu_{\mathcal{B}}} \alpha^{m,\nu}_{\boldsymbol{j},\boldsymbol{t}}\, b^\nu_{\boldsymbol{t}} \qquad \forall\, (m, \boldsymbol{j}) \in \mathcal{I}_{\mathcal{H}} \quad \text{such that} \quad \operatorname{supp}(b^\ell_{\boldsymbol{i}}) \cap \operatorname{supp}(b^m_{\boldsymbol{j}}) \neq \emptyset, \tag{10}$$

with $\alpha^{m,\nu}_{\boldsymbol{j},\boldsymbol{t}} > 0$ iff $\operatorname{supp}(b^\nu_{\boldsymbol{t}}) \subseteq \operatorname{supp}(b^m_{\boldsymbol{j}})$ and $\alpha^{m,\nu}_{\boldsymbol{j},\boldsymbol{t}} = 0$ otherwise.

In order to define the quadrature rule $\mathfrak{Q}^\ell_{\boldsymbol{i}}$ for the hierarchical space $\mathcal{H}$, we rely on the definitions and relations introduced in the previous section for tensor-product spaces. In (7) we take

$$\mathcal{I}^{(\ell,\boldsymbol{i})}_{\mathcal{Q}} \coloneqq \left\{ \mathbf{q} \in \mathcal{I}^\nu_{\mathcal{Q}} \;:\; \bar{\boldsymbol{x}}^\nu_{\mathbf{q}} \in \mathcal{Q}^\nu \cap \operatorname{supp}(b^\ell_{\boldsymbol{i}}) \right\}$$

and let the quadrature points be $\boldsymbol{x}^\ell_{\boldsymbol{i},\mathbf{q}} = \bar{\boldsymbol{x}}^\nu_{\mathbf{q}}$ for every $\mathbf{q} \in \mathcal{I}^{(\ell,\boldsymbol{i})}_{\mathcal{Q}}$, hence (from definition (6)) we have

$$\mathcal{Q}^{(\ell,\boldsymbol{i})} = \mathcal{Q}^\nu \cap \operatorname{supp}(b^\ell_{\boldsymbol{i}})\,. \tag{11}$$

See Figure 1 for an example of quadrature points for a basis function $b^1_{\boldsymbol{i}} \in \mathcal{H}$. Since it interacts with a level 2 basis function and not with level 3 one, it inherits a local set of level 2 quadrature points $\bar{\boldsymbol{x}}^2_{\mathbf{q}}$.

The quadrature weights for $b^\ell_{\boldsymbol{i}}$ are simply

$$w^\ell_{\boldsymbol{i},\mathbf{q}} = \sum_{\boldsymbol{s} \in \mathcal{I}^\nu_{\mathcal{B}}} \alpha^{\ell,\nu}_{\boldsymbol{i},\boldsymbol{s}} \bar{w}^\nu_{\boldsymbol{s},\mathbf{q}} \tag{12}$$

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 (10).

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* (8) *on the hierarchical space.*

Proof. Let $(m,\boldsymbol{j}) \in \mathcal{I}_{\mathcal{H}}$. If $\operatorname{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \operatorname{supp}(b_{\boldsymbol{j}}^{m}) = \emptyset$, then the equation in (8) 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 $\operatorname{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \operatorname{supp}(b_{\boldsymbol{j}}^{m}) \neq \emptyset$, then

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

In the penultimate row 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 $\left\{\bar{w}_{\boldsymbol{s},\mathbf{q}}^{\nu}\right\}_{\mathbf{q}\in\mathcal{I}_{\mathcal{Q}}^{\nu}}$ satisfy the exactness conditions (4) on the level $\nu = \nu(\ell,\boldsymbol{i})$. □

Remark 1. *Quadrature rule $\mathfrak{Q}_{\boldsymbol{i}}^{\ell}$ in* (7) *of level ℓ is actually a linear combination of quadrature rules defined at level ν. Namely, if $\operatorname{supp}(b_{\boldsymbol{i}}^{\ell}) \cap \operatorname{supp}(v) \neq \emptyset$ we can define $\mathcal{I} := \{\boldsymbol{s} \in \mathcal{I}_{\mathcal{B}}^{\nu} \,:\, \operatorname{supp}(b_{\boldsymbol{s}}^{\nu}) \subseteq \operatorname{supp}(b_{\boldsymbol{i}}^{\ell})\}$ and derive*

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

In particular, the quadrature rule $\mathfrak{Q}^\ell_{\boldsymbol{i}}$ is determined by those rules $\bar{\mathfrak{Q}}^\nu_{\boldsymbol{s}}$ whose $\mathrm{supp}(b^\nu_{\boldsymbol{s}})$ *are in the support of $b^\ell_{\boldsymbol{i}}$.*

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 $b^\ell_{\boldsymbol{i}}$ and of the quadrature points for the full level ℓ basis functions. Quadrature weights are therefore obtained in two steps. First, we compute the univariate quadrature weights of level ν, defined in (9), 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 $b^\ell_{\boldsymbol{i}}$ are computed as linear combination of level ν quadrature weights, analogously to (12).

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 ones for each level, i.e., we classify the basis functions of $\mathcal{H}$ with respect to ν by defining the *s*ets of level n interacting functions

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

It is trivial to check the following properties:

$$\mathcal{H} = \bigcup_{n \le L} F^n, \tag{13}$$

$$F^m \cap F^n = \emptyset \text{ if } m \neq n, \tag{14}$$

$$F^n \text{ only contains functions of } \mathcal{H} \text{ of levels } \le n \, . \tag{15}$$

The classification of the active basis function $b^\ell_{\boldsymbol{i}}$ 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**
  $F^{\nu(\ell,\boldsymbol{i})} = F^{\nu(\ell,\boldsymbol{i})} \cup b^\ell_{\boldsymbol{i}}$
end foreach

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

Because of (10) and (15), every function $b^\ell_{\boldsymbol{i}} \in F^n$ can be written as linear combination of functions $b^n_{\boldsymbol{j}}$ from level n, and a similar formula holds for its quadrature weights (see (12)). Analogously, the same can be said for the (univariate) components: b^ℓ_{k,i_k} can

be written as a linear combination of level n functions b^n_{k,j_k}, and w^ℓ_{k,i_k,q_k} as a linear combination of $\bar{w}^n_{k,j_k,q_k}$,

$$b^\ell_{k,i_k} = \sum_{j_k \in D^{\ell,n}_{k,i_k}} \alpha^{\ell,n}_{k,i_k,j_k}\, b^n_{k,j_k}, \qquad w^\ell_{k,i_k,q_k} = \sum_{j_k \in D^{\ell,n}_{k,i_k}} \alpha^{\ell,n}_{k,i_k,j_k} \bar{w}^n_{k,j_k,q_k} \tag{16}$$

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

$$D^{\ell,n}_{k,i_k} = \{j_k \in \{1, \ldots, N^n_{\mathcal{B}_k}\} \ : \ \mathrm{supp}(b^n_{k,j_k}) \subseteq \mathrm{supp}(b^\ell_{k,i_k})\}.$$

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

$$\pi_k \boldsymbol{i} \coloneqq i_k, \qquad \tau_k \boldsymbol{i} \coloneqq (i_1, \ldots, i_k),$$

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

$$G^{\ell,n}_k \coloneqq \{i_k = \pi_k \boldsymbol{i} \ : \ b^\ell_{\boldsymbol{i}} \in \mathcal{H}^\ell \cap F^n\}. \tag{17}$$

Finally, we define

$$D^n_k \coloneqq \bigcup_{\ell=0}^{n} \bigcup_{i_k \in G^{\ell,n}_k} D^{\ell,n}_{k,i_k} \tag{18}$$

The sets D^n_k identify the univariate quadrature weights that are needed to set up the WQ rule. Precisely, in the next step for each $j_k \in D^n_k$ we compute the nonzero univariate quadrature weights $\{\bar{w}^n_{k,j_k,q_k}\}_{q_k=1}^{R^n_k}$ associated to $b^n_{k,j_k} \in \mathcal{B}^n_k$ by imposing the univariate exactness conditions analogous to (4), exactly as done in [11]. Namely, we impose that

$$\bar{w}^n_{k,j_k,q_k} = 0 \qquad \text{if } \bar{x}^n_{k,q_k} \notin \mathrm{supp}(b^n_{k,j_k}),$$

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

$$\sum_{q_k \in A^{n,n}_{k,j_k}} \bar{w}^n_{k,j_k,q_k} b^n_{k,t_k}(\bar{x}^n_{k,q_k}) = \int_0^1 b^n_{k,j_k}(\xi)\, b^n_{k,t_k}(\xi)\, d\xi, \qquad b^n_{k,t_k} \in B^n_{k,j_k}, \tag{19}$$

where

$$A^{\ell,n}_{k,j_k} \coloneqq \{q_k \in \{1, \ldots, R^n_k\} \ : \ \bar{x}^n_{k,q_k} \in \mathcal{Q}^n_k \cap \mathrm{supp}(b^\ell_{k,j_k})\}, \tag{20}$$

$$B^n_{k,j_k} \coloneqq \{b^n_{k,t_k} \in \mathcal{B}^n_k \ : \ \mathrm{supp}(b^n_{k,j_k}) \cap \mathrm{supp}(b^n_{k,t_k}) \neq \emptyset\} \tag{21}$$

are the set of indices of quadrature points inside the support of the basis function b^ℓ_{k,j_k}, and the corresponding interacting trial univariate functions, respectively. The construction of the univariate quadrature weights is summarized in Algorithm 2.

Up to this point, we have defined the univariate quadrature points and computed the univariate quadrature weights, associated to all the basis functions in $\mathcal{B}^n_k$ that are needed

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 (20)

find the interacting trial univariate functions B_{k,j_k}^n from (21)

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 (19)

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

to represent functions in F^n as linear combination of functions of level n, by using (10). Using (16) we can then compute the level ℓ univariate quadrature weights

$$\mathcal{W}_{k,i_k}^{\ell,n} \coloneqq \left\{ w_{k,i_k,q_k}^{\ell} = \sum_{j_k \in D_{k,i_k}^{\ell,n}} \alpha_{k,i_k,j_k}^{\ell,n} \bar{w}_{k,j_k,q_k}^n \ : \ q_k \in A_{k,i_k}^{\ell,n} \right\} \qquad k = 1, \dots, d \,, \quad (22)$$

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,

$$\Psi^n \coloneqq \bigcup_{b_{\boldsymbol{i}}^{\ell} \in F^n} \mathrm{supp}(b_{\boldsymbol{i}}^{\ell}), \quad (23)$$

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

$$\mathcal{Q}_{\Psi}^n \coloneqq \mathcal{Q}^n \cap \Psi^n. \quad (24)$$

Remark 2. *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, $\mathcal{Q}_{\Psi} \coloneqq \bigcup_{n \le L} \mathcal{Q}_{\Psi}^n$, that is used for the evaluation of the non-tensor product coefficients.*

The complete preprocessing phase is described by the Algorithm 3.

3.3. Matrix formation: algorithm

The rows and the 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:

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

Algorithm 3: `preprocessing`

Input : $\mathcal{H}$

$\{F^n\}_{n=0}^{L} =$ `classify_basis_functions(`$\mathcal{H}$`)` (Alg. 1)
for $n = 0, \ldots, L$ *such that* $F^n \neq \emptyset$ **do**
  for $k = 1, \ldots, d$ **do**
    compute 1D quadrature points $\mathcal{Q}_k^n = \{\bar{x}_{k,q_k}^n\}_{q_k=1}^{R^n}$ for $\mathcal{B}_k^n$ (see Section 2.2)
    for $\ell = 0, \ldots, n$ **do**
      compute $G_k^{\ell,n}$ from (17)
    end for
    compute D_k^n from (18)
    foreach $j_k \in D_k^n$ **do**
      $\{\bar{w}_{k,j_k,q_k}^n\}_{q_k \in A_{k,j_k}^{n,n}} =$ `compute_1Dweights(`$\mathcal{B}_k^n$, j_k, $\mathcal{Q}_k^n$`)` (Alg. 2)
    end foreach
    $\mathcal{W}_k^n = \emptyset$
    for $\ell = 0, \ldots, n$ **do**
      foreach $i_k \in G_k^{\ell,n}$ **do**
        compute weights $\mathcal{W}_{k,i_k}^{\ell,n}$ from (22)
        $\mathcal{W}_k^n = \mathcal{W}_k^n \cup \mathcal{W}_{k,i_k}^{\ell,n}$
      end foreach
    end for
  end for
  compute the d-dimensional tensor-product points $\mathcal{Q}^n$ from (5)
  compute Ψ^n from (23)
  compute $\mathcal{Q}_\Psi^n$ from (24)
end for

Output: $F^n, \mathcal{W}_1^n, \ldots, \mathcal{W}_d^n, \mathcal{Q}^n, \mathcal{Q}_\Psi^n$, for $n = 0, \ldots, L$

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 (7), and the fact that $w^\ell_{\boldsymbol{i},\mathbf{q}} = 0$ for $\bar{\boldsymbol{x}}^n_{\mathbf{q}} \notin \operatorname{supp}(b^\ell_{\boldsymbol{i}})$, we can write

$$[M]_{(\ell,\boldsymbol{i}),(m,\boldsymbol{j})} \approx \mathfrak{Q}^\ell_{\boldsymbol{i}}(c\, b^m_{\boldsymbol{j}}) = \sum_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}} w^\ell_{\boldsymbol{i},\mathbf{q}} c(\bar{\boldsymbol{x}}^n_{\mathbf{q}}) b^m_{\boldsymbol{j}}(\bar{\boldsymbol{x}}^n_{\mathbf{q}})\, d\boldsymbol{x}, \tag{25}$$

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 (25) in terms of nested sums:

$$\begin{aligned}\mathfrak{Q}^\ell_{\boldsymbol{i}}(c b^m_{\boldsymbol{j}}) &= \sum_{\mathrm{q}_1,\dots,\mathrm{q}_d} \prod_{k=1}^d \left(w^\ell_{k,i_k,\mathrm{q}_k} b^m_{k,j_k}(\bar{x}^n_{k,\mathrm{q}_k})\right)\, c(\bar{x}^n_{1,\mathrm{q}_1},\dots,\bar{x}^n_{d,\mathrm{q}_d}) \\ &= \sum_{\mathrm{q}_d} w^\ell_{d,i_d,\mathrm{q}_d} b^m_{d,j_d}(\bar{x}^n_{d,\mathrm{q}_d}) \left(\sum_{\mathrm{q}_{d-1}} \cdots \sum_{\mathrm{q}_1} w^\ell_{1,i_1,\mathrm{q}_1} b^m_{1,j_1}(\bar{x}^n_{1,\mathrm{q}_1}) c(\bar{x}^n_{1,\mathrm{q}_1},\dots,\bar{x}^n_{d,\mathrm{q}_d}) \right)\end{aligned} \tag{26}$$

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

$$Q^{n,\ell,m}_{k,i_k,j_k} \coloneqq \{\mathrm{q}_k \in \{1,\dots,R^n_k\}\colon \bar{x}^n_{k,\mathrm{q}_k} \in \operatorname{supp}(b^\ell_{k,i_k}) \cap \operatorname{supp}(b^m_{k,j_k})\}\,. \tag{27}$$

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 (26) we note that coefficient $c(\boldsymbol{x})$ must be evaluated at the points of the quadrature rule of level n. Moreover, from (13)–(15) 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:

$$C^n_{\mathbf{q}} = C^n_{(\mathrm{q}_1,\dots,\mathrm{q}_d)} \coloneqq \begin{cases} c(\bar{\boldsymbol{x}}^n_{\mathbf{q}}) & \text{if } \bar{\boldsymbol{x}}^n_{\mathbf{q}} \in \mathcal{Q}^n_\Psi \\ 0 & \text{otherwise} \end{cases}. \tag{28}$$

Remark 3. *The coefficients $C^n_{\mathbf{q}}$ will be used in the innermost loop of the sum-factorization algorithm, so they should be stored in an efficient data structure for the data retrieval w.r.t. the loop index ordering used in the sum-factorization.*

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.,

$$K^n_{\ell,m} \coloneqq \{(\boldsymbol{i},\boldsymbol{j}) \in \mathcal{I}^\ell_{\mathcal{B}} \times \mathcal{I}^m_{\mathcal{B}} \;:\; b^\ell_{\boldsymbol{i}} \in \mathcal{H}^\ell \cap F^n\,,\; b^m_{\boldsymbol{j}} \in \mathcal{H}^m\,,\; \operatorname{supp}(b^\ell_{\boldsymbol{i}}) \cap \operatorname{supp}(b^m_{\boldsymbol{j}}) \neq \emptyset\}\,. \tag{29}$$

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

The sum-factorization algorithm in essence is a clever way to perform the nested sum (26), 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.

The integration along direction $k = 1$ writes as

$$\begin{aligned} I^{(1)}_{(i_1),(j_1);(\mathrm{q}_2,\dots,\mathrm{q}_d)} &\coloneqq \sum_{\mathrm{q}_1} w^{\ell}_{1,i_1,\mathrm{q}_1} b^{m}_{1,j_1}(\bar{x}^{n}_{1,q_1}) C^{n}_{(\mathrm{q}_1,\dots,\mathrm{q}_d)} \\ &= \sum_{\mathrm{q}_1} w^{\ell}_{1,i_1,\mathrm{q}_1} b^{m}_{1,j_1}(\bar{x}^{n}_{1,q_1}) I^{(0)}_{(),();(\mathrm{q}_1,\dots,\mathrm{q}_d)} \ , \end{aligned} \tag{30}$$

where we have defined $I^{(0)}_{(),();(\mathrm{q}_1,\dots,\mathrm{q}_d)} \equiv C^{n}_{(\mathrm{q}_1,\dots,\mathrm{q}_d)}$, 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:

$$I^{(k)}_{(i_1,\dots,i_k),(j_1,\dots,j_k);(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)} \coloneqq \sum_{\mathrm{q}_k} w^{\ell}_{k,i_k,\mathrm{q}_k} b^{m}_{k,j_k}(\bar{x}^{n}_{k,q_k}) I^{(k-1)}_{(i_1,\dots,i_{k-1}),(j_1,\dots,j_{k-1});(\mathrm{q}_k,\dots,\mathrm{q}_d)} \ , \tag{31}$$

and finally for $k = d$:

$$I^{(d)}_{(i_1,\dots,i_d),(j_1,\dots,j_d);()} \coloneqq \sum_{\mathrm{q}_d} w^{\ell}_{d,i_d,\mathrm{q}_d} b^{m}_{d,j_d}(\bar{x}^{n}_{d,q_d}) I^{(d-1)}_{(i_1,\dots,i_{d-1}),(j_1,\dots,j_{d-1});(\mathrm{q}_d)} = \mathfrak{Q}^{\ell}_{\boldsymbol{i}}(cb^{m}_{\boldsymbol{j}}) \ , \tag{32}$$

where the final expression in (32) 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 $\mathfrak{Q}^{\ell}_{\boldsymbol{i}}(cb^{m}_{\boldsymbol{j}})$.

The key point (which allows to save computations) is 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 (31) 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:

$$\Pi^{(k)} K^{n}_{\ell,m} \coloneqq \left\{ \big((i_1, \dots, i_k), (j_1, \dots, j_k)\big) = \big(\tau_k \boldsymbol{i}, \tau_k \boldsymbol{j}\big) \ , \forall \big(\boldsymbol{i}, \boldsymbol{j}\big) \in K^{n}_{\ell,m} \right\} \tag{33}$$

and then the pairs of k-tuple that must be considered for the efficient computation of (31) are just the elements of $\Pi^{(k)} K^{n}_{\ell,m}$.

The sum-factorization algorithm is then summarized by Algorithm 4.

The algorithm for the matrix formation is depicted by Algorithm 5.

Algorithm 4: `sum_factorization`

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

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 (33)
  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 $Q^{n,\ell,m}_{k,i_k,j_k}$ from (27);
  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 (22))
    foreach $(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)\in\{1,\dots,R^n_{k+1}\}\times\cdots\times\{1,\dots,R^n_d\}$ **do**
      compute $I^{(k)}_{(i_1,\dots,i_k),(j_1,\dots,j_k);(\mathrm{q}_{k+1},\dots,\mathrm{q}_d)}$ from (31)
    end foreach
  end foreach
end for

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

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 $\{C^n_{\mathbf{q}}\}_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}}$ from (28)
  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 (29)
      $\{[M]_{(\ell,\boldsymbol{i}),(m,\boldsymbol{j})}\}_{(\boldsymbol{i},\boldsymbol{j})\in K^n_{\ell,m}} =$
       `sum_factorization`$(\mathcal{I}^n_{\mathcal{Q}},\{C^n_{\mathbf{q}}\}_{\mathbf{q}\in\mathcal{I}^n_{\mathcal{Q}}},\ell,K^n_{\ell,m},\{\mathcal{W}^n_k\}_{k=1}^d)$ (Alg. 4)
    end foreach
  end foreach
end foreach

Output: M

4. Computational cost

We now want to estimate the total computational cost of the matrix formation. 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 dominated from the interior part in all cases of interest. Recalling (11) and $\nu - \ell \leq r - 1$, the number of quadrature points that belong to interior elements is bounded by

$$\sum_{(\ell,\boldsymbol{i})\in\mathcal{I}_{\mathcal{H}}} \#\mathcal{Q}^{(\ell,\boldsymbol{i})} \leq \sum_{(\ell,\boldsymbol{i})\in\mathcal{I}_{\mathcal{H}}} (2^{r-1}(p+1))^d = O(2^{dr}p^d N_{\mathcal{H}}). \tag{34}$$

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 (19) 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 nonzero weights 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 related to the computation of 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 now 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 (25) 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 (26) 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 $\bar{x}^n_{k,\mathrm{q}_k}$ 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. Typically, the cost is dominated by the quadrature at the interior, therefore we assume that the number of index values taken by q_k in the k-th sum of (26) is roughly $2^{n-\ell+1}(p + 1)$.

We are now ready to estimate the cost of computing (26). As a first step, we evaluate the innermost sum (30) 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 nonzero terms, and we observe that the term corresponding to a fixed q_1 is nonzero only for the $p+1$ values of the index 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 compution of the sum (30) requires 2 flops for each of its nonzero terms.

Since each index $\mathrm{q}_1, \ldots, \mathrm{q}_d$, can take up to $2^{n-\ell+1}(p+1)$ values, and 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

$$2\,(p+1)^{d+1}\; 2^{d(n-\ell+1)} N_{n,\ell} \text{ flops.} \tag{35}$$

where

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

For $k = 2, \ldots, d-1$, the k-th step of the sum-factorization requires the computation of (31) for all values of $i_1, \ldots, i_k$, $j_1, \ldots, j_k$ and $\mathrm{q}_{k+1}, \ldots, \mathrm{q}_d$, where the inner sum $I^{(k-1)}_{(i_1,\ldots,i_{k-1}),(j_1,\ldots,j_{k-1});(\mathrm{q}_k,\ldots,\mathrm{q}_d)}$ has already been computed for all the relevant index values.

Since $(i_1, \ldots, i_k) = \tau_k \boldsymbol{i}$ the total number of k-tuples $(i_1, \ldots, i_k)$ 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 $\mathrm{q}_k, \ldots, \mathrm{q}_d$ and $j_1, \ldots, j_{k-1}$ is bounded by $2^{n-\ell+1}(p+1)$, the cost of this step is again bounded by (35). With similar arguments, it can be shown that this is true also for the $d-$th step of the sum-factorization (31).

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

$$2d(2r-1)\,(p+1)^d \sum_n \sum_{\ell \leq n} 2^{d(n-\ell+1)} N_{n,\ell} \text{ flops.} \tag{36}$$

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

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

Hence the total cost for the matrix entries computation (36) is bounded by

$$2d(2r-1)2^{dr}\,(p+1)^{d+1}\, N_{\mathcal{H}} = O\left(dr 2^{dr} p^{d+1} N_{\mathcal{H}}\right) \text{ flops.} \tag{37}$$

We observe that, similarly as in the bound on the active quadrature points (34), the latter expression grows exponentially with respct to the admissibilty parameter r, and this effect worsen with the increasing of the dimension d. This might seem unsatisfactory, but we emphasize that (37) is easily a rather pessimistic bound. Indeed, a careful analysis

of the derivation of (37) reveals that we are essentially assuming that every hierarchical B-spline basis function $b_{\boldsymbol{i}}^{\ell}$, with $(\ell, \boldsymbol{i}) \in \mathcal{I}_{\mathcal{H}}$, interacts with all the admissible levels. In many practical cases, however, refinement is perfomred only in specific regions of the domain, e.g., in the neighbourhood of low dimensional manifolds, and as a result the number of basis functions that interact with all the admissible levels is limited.

5. Numerical tests

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

$$f(\mathbf{x}) = \exp\left[-\Big(\frac{\|\mathbf{x} - \mathbf{x}_0\| - 1}{\beta}\Big)^2\right], \tag{38}$$

where the physical domain $\Omega \in \mathbb{R}^d$, the parameter $\beta \in \mathbb{R}$ and the point $\mathbf{x}_0 \in \mathbb{R}^d$ are specified below for the cases $d = 2$ and $d = 3$, using a r-admissible hierarchical B-spline basis (see Section 2.1) of degree p, with different values for the admissibility parameter r.

For each value of the admissibility parameter r, a nested sequence of hierarchical B-spline spaces is constructed [28]. The adaptive mesh refinement is steered by the "error estimator", which is simply the L^2-error between the computed L^2 projection and the function (38) and by using the Dörfler marking strategy [29] with parameter $\theta_* = 0.2$.

For each refinement step, we perform a simulation using the standard element-base Gaussian quadrature (using $p + 1$ quadrature point along each direction of the element) to build the mass matrix (and right hand side) and then, using the same sequence of hierarchical spaces we compute the mass-matrix using the proposed hierarchical WQ algorithm.

Remark 4. *All the numerical tests were performed using the IGATOOLS library [30], on a single core of an Intel Xeon E5-2470 processor running at 2.3 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 5. *In all plots the lower limit of the CPU time is set to* 10^{-1} *seconds to reduce the effect of random time fluctuations, due to the CPU scheduling.*

5.1. 2D case

For this case the physical domain $\Omega \subset \mathbb{R}^2$ 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

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

while $\beta = 5 \cdot 10^{-3}$ and $\mathbf{x}_0 = (0, \frac{5}{2})$ (see Figure 2).

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

Regarding the cases with $r = 2$ we can observe from the plots in Figure 3 that the total time (preprocessing + matrix computation) w.r.t. $N_{\mathcal{H}}$ or the WQ approach seems to

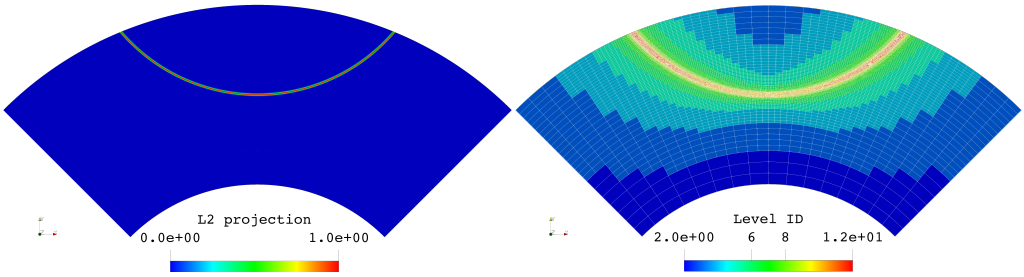


Figure 2: L^2-projection (left) and element levels (right) after 51 adaptive refinements for the 2D case with degree $p = 2$ and admissibility $r = 2$. For this configuration the space contains 867947 degrees of freedom, and the L^2-error between the function (38) and its L^2-projection is $\approx$ 3.4e−7.

be nearly independent from 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 (except for some specific space configurations when $p = 3$). 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 Figure 4).

Regarding the CPU cost of the WQ approach, in Figure 5 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 (28) (for $n = 0, \dots, L$) and the rest of the algorithm (i.e., the computation of the connectivities $K^n_{\ell,m}$ from (29) and the sum-factorization). From the plots in Figure 5 we have that the asymptotic behaviour of the costs is the same for all different degrees, resulting in the dominant cost being the formation of the matrix whereas the 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. Moreover, the cost for evaluating the coefficients in in Eq. (28) depends on the number of points in $\{\mathcal{Q}^n_\Psi\}_{n=0}^L$ and che cost of evaluation of the function c at a single point. In our case c is just the determinant of the Jacobian of the mapping, resulting in low CPU time w.r.t. the other two main costs.

Regarding the case with admissibility parameter $r = 3$, we observe from the plots in Figure 6 that both approaches (element-based Gaussian quadrature and WQ) have an higher cost (for a given number of dofs) for all tested degrees w.r.t. the case with $r = 2$ (Figure 3), but the WQ approach seems to be nearly independent of the degree and it less expensive w.r.t. the element-based Gaussian approach of degree >= 3.

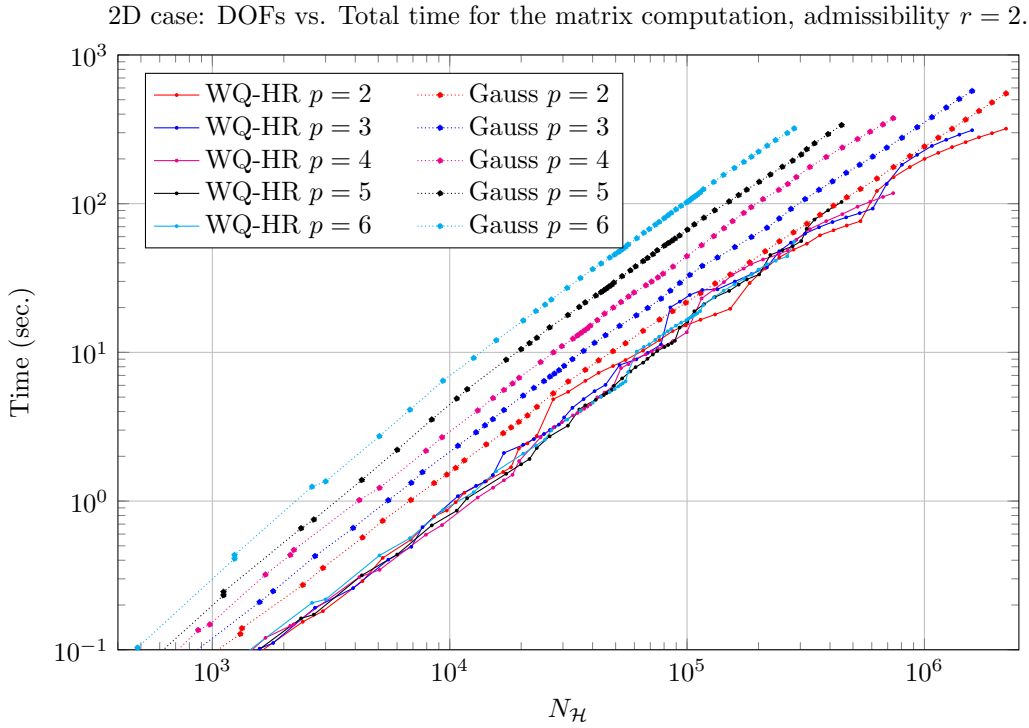


Figure 3: $N_{\mathcal{H}}$ vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 2$. For WQ-HR the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

5.2. 3D case

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

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

while $\beta = 0.1$ and $\mathbf{x}_0 = (0, \frac{5}{2}, 0)$ (see the Figure 7).

For this case we performed simulations using the admissibility parameter $r = 2$ and the degrees $p = 2, \ldots, 5$.

We can observe from the plots in Figures 8 and 9 that the WQ approach outperforms the element-based Gaussian approach. In fact, considering the CPU time w.r.t. $N_{\mathcal{H}}$, 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 have that if one want achieve a low error level, the best strategy in terms of CPU time needed to build the matrix is to use WQ with high degree (see Figure 9).

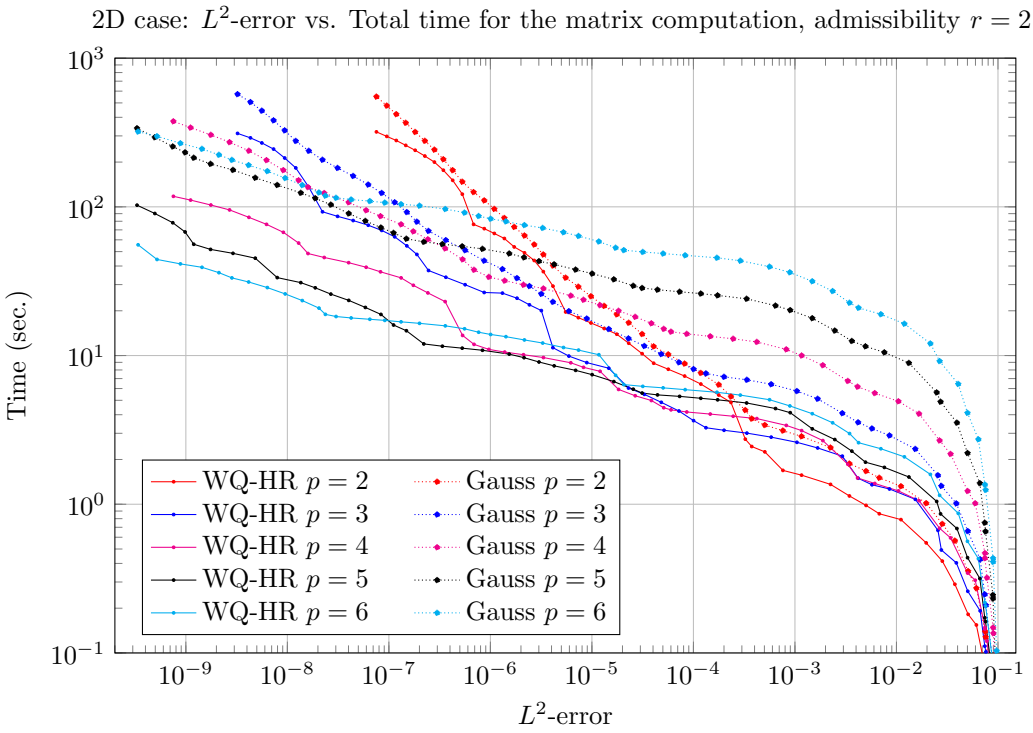


Figure 4: L^2-error vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 2$. For WQ-HR the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

Regarding the CPU cost of the WQ approach, in Figure 10 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 Eq. (28) (for $n = 0, \ldots, L$) and the rest of the algorithm (i.e. the computation of the connectivities $K^n_{\ell,m}$ from (29) and the sum-factorization). In this case, w.r.t. the 2D case we observe that for the degree $p = 2$, when we have a number of degrees of freedom $< 1.5\text{e}5$, the dominant cost can be attributed to the evaluation of the coefficients in (28) whereas for an higher number of degrees of freedom, the dominant cost is due to the sum-factorization+computation of the connectivities $K^n_{\ell,m}$ phase. When we increase the degree we observe that this cost becomes the dominant one when $p > 3$ (and conversely the cost for the coefficients evaluation becomes the lowest of the three costs for $p > 3$).

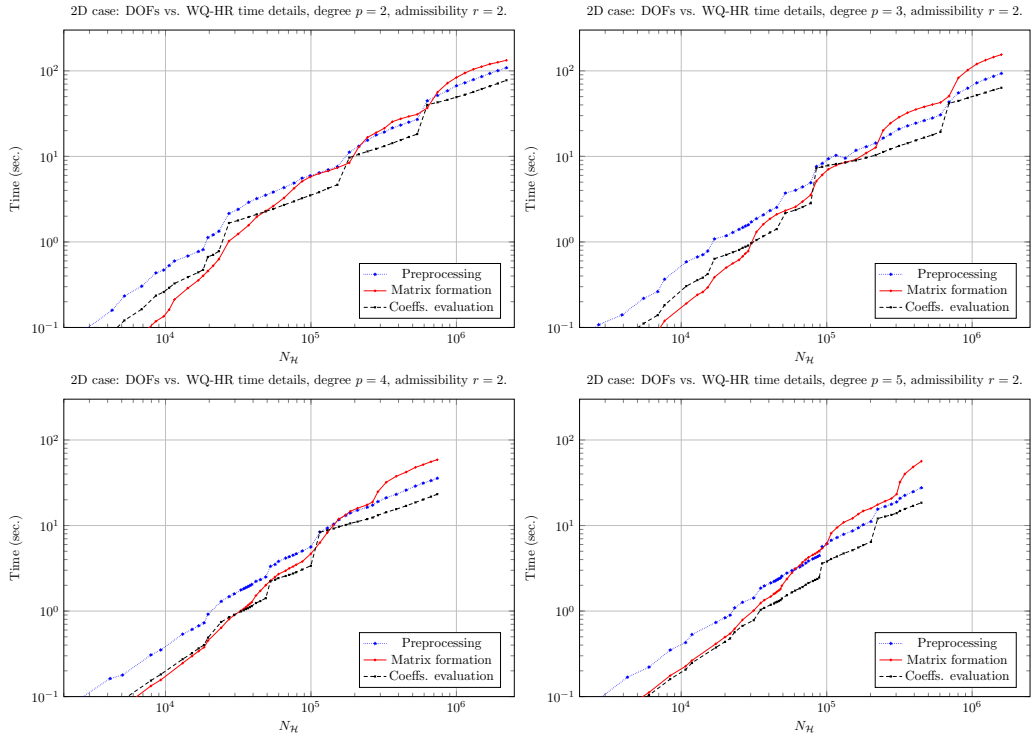


Figure 5: $N_{\mathcal{H}}$ vs. the CPU time needed to run the WQ-HR algorithm: preprocessing (Algorithm 3) and matrix computation (Algorithm 5) for the 2D case and admissibility parameter $r = 2$. The cost for the matrix computation is split in the CPU cost for coefficient evaluations ((28) for $n = 0, \dots, L$) and the CPU cost for executing the rest of the Algorithm 5.

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. 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 the combination of the proposed algorithm with matrix-free methods [31] as well as the extension to the case of truncated hierarchical B-splines [3, 4], and the application to PDE problems of applicative interest.

References

[1] A. Buffa, G. Gantner, C. Giannelli, D. Praetorius, R. Vázquez, Mathematical foundations of adaptive isogeometric analysis (2021). 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.

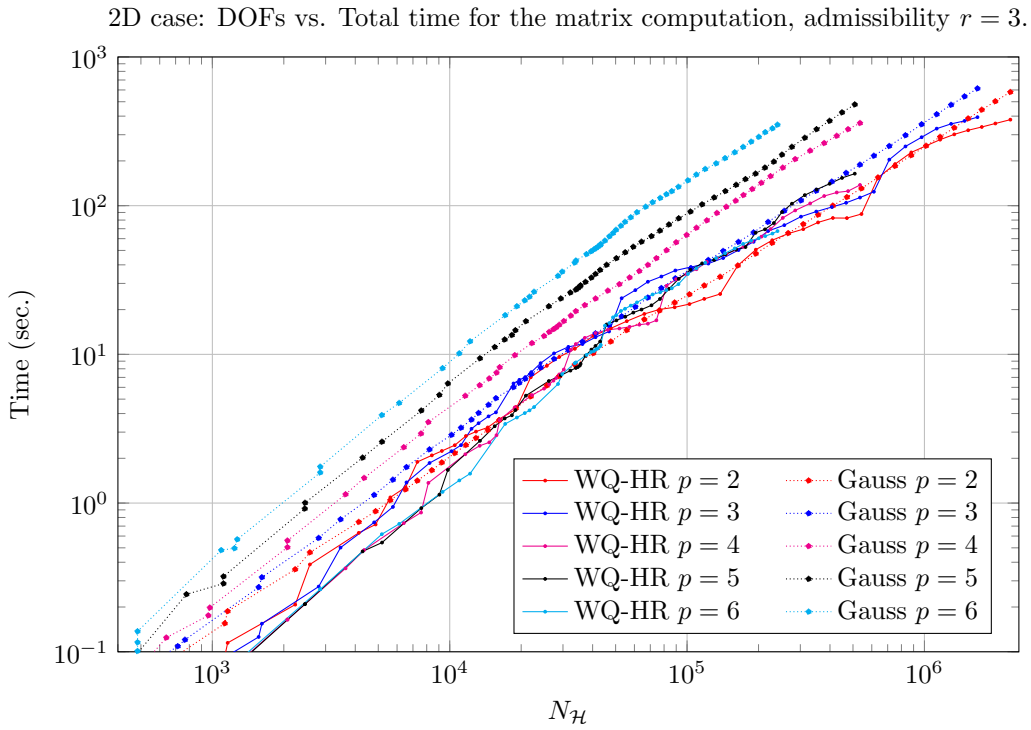


Figure 6: DOFs vs. total time for the matrix computation for the 2D case and admissibility parameter $r = 3$. For WQ-HR the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

[3] C. Giannelli, B. Jüttler, H. Speleers, THB-splines: The truncated basis for hierarchical splines, Comput. Aided Geom. Design 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. arXiv:1701.07764.

[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, Computer Methods in Applied Mechanics and Engineering 366 (2020)

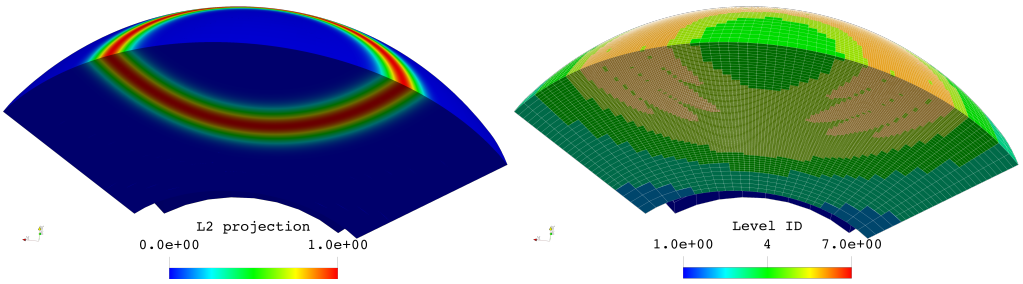


Figure 7: L^2-projection (left) and element levels (right) after 27 adaptive refinements for the 3D case with degree $p = 2$ and admissibility $r = 2$. For this configuration the space contains 754614 dofs, and the L^2-error between the function (38) and its L^2-projection is $\approx$ 8.5e−6. .

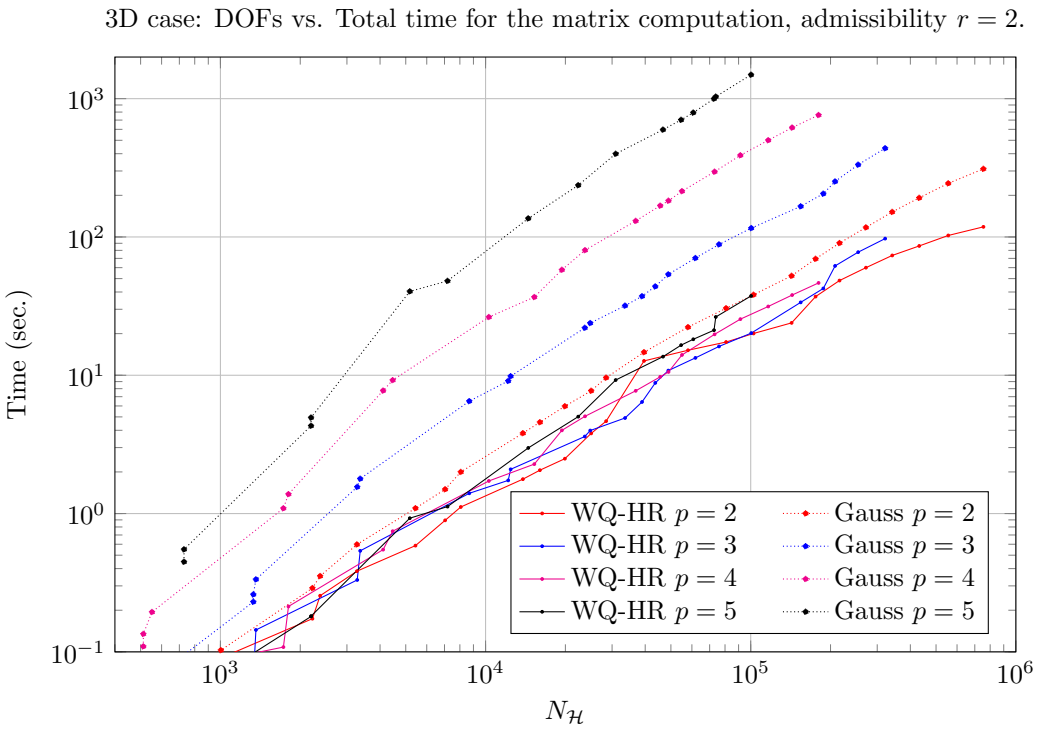


Figure 8: DOFs vs. total time for the matrix computation for the 3D case and admissibility parameter $r = 2$. For WQ-HR the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

113005.

[13] M. Pan, B. Jüttler, A. Mantzaflaris, Efficient matrix assembly in isogeometric analysis with hierarchical b-splines, Journal of Computational and Applied Mathematics 390 (2021) 113278.

[14] T. Hirschler, P. Antolin, A. Buffa, Fast and multiscale formation of isogeometric matrices of microstructured geometric models, arXiv preprint arXiv:2107.09568.

[15] P. Antolin, A. Buffa, F. Calabro, M. Martinelli, G. Sangalli, Efficient matrix computation for

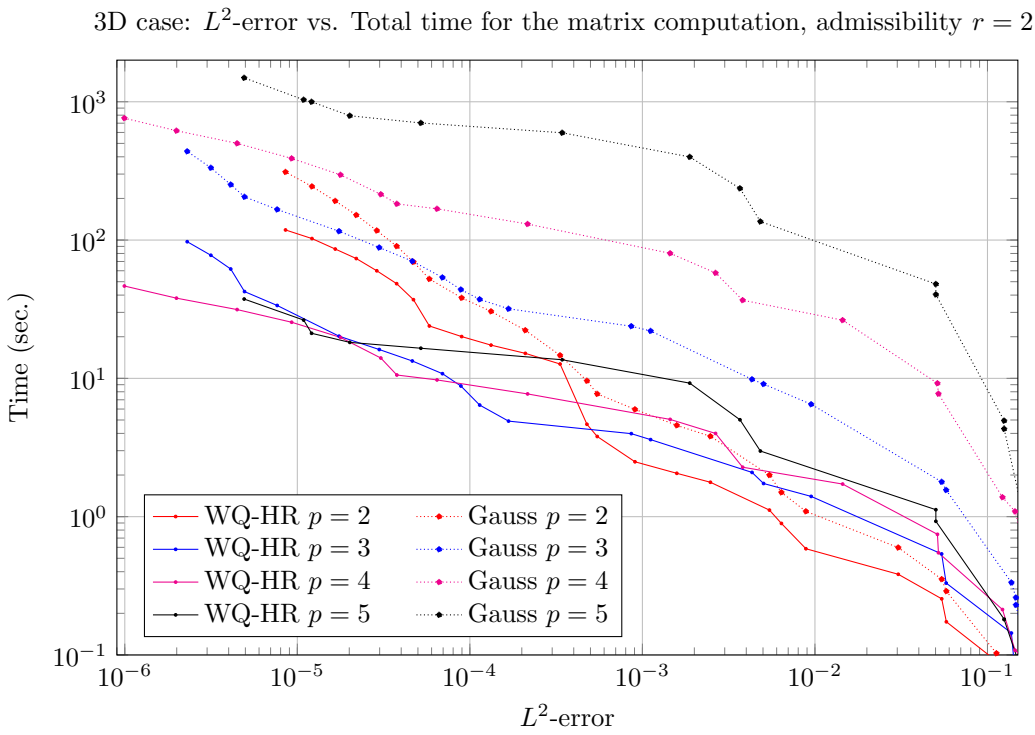


Figure 9: L^2-error vs. total time for the matrix computation for the 3D case and admissibility parameter $r = 2$. For WQ-HR the total time for the matrix computation is the sum of the time needed for the preprocessing (Section 3.2) and for the matrix formation (Section 3.3).

tensor-product isogeometric analysis: The use of sum factorization, Computer Methods in Applied Mechanics and Engineering 285 (2015) 817–828.

[16] A. Bressan, S. Takacs, Sum factorization techniques in isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 352 (2019) 437–460.

[17] D. Drzisga, B. Keith, B. Wohlmuth, The surrogate matrix methodology: Accelerating isogeometric analysis of waves, Computer Methods in Applied Mechanics and Engineering 372 (2020) 113322.

[18] F. Fahrendorf, L. De Lorenzis, H. Gomez, Reduced integration at superconvergent points in isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 328 (2018) 390–410.

[19] A. Mantzaflaris, B. Jüttler, B. N. Khoromskij, U. Langer, Low rank tensor methods in Galerkin-based isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 316 (2017) 1062–1085.

[20] A. Karatarakis, P. Karakitsios, M. Papadrakakis, Gpu accelerated computation of the isogeometric analysis stiffness matrix, Computer Methods in Applied Mechanics and Engineering 269 (2014) 334–355.

[21] T. J. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Computer methods in applied mechanics and engineering 199 (5-8) (2010) 301–313.

[22] 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, Computer Methods in Applied Mechanics and Engineering 249 (2012) 15–27.

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

[24] M. Bartoň, V. M. Calo, Optimal quadrature rules for odd-degree spline spaces and their application to tensor-product-based isogeometric analysis, Computer Methods in Applied Mechanics and

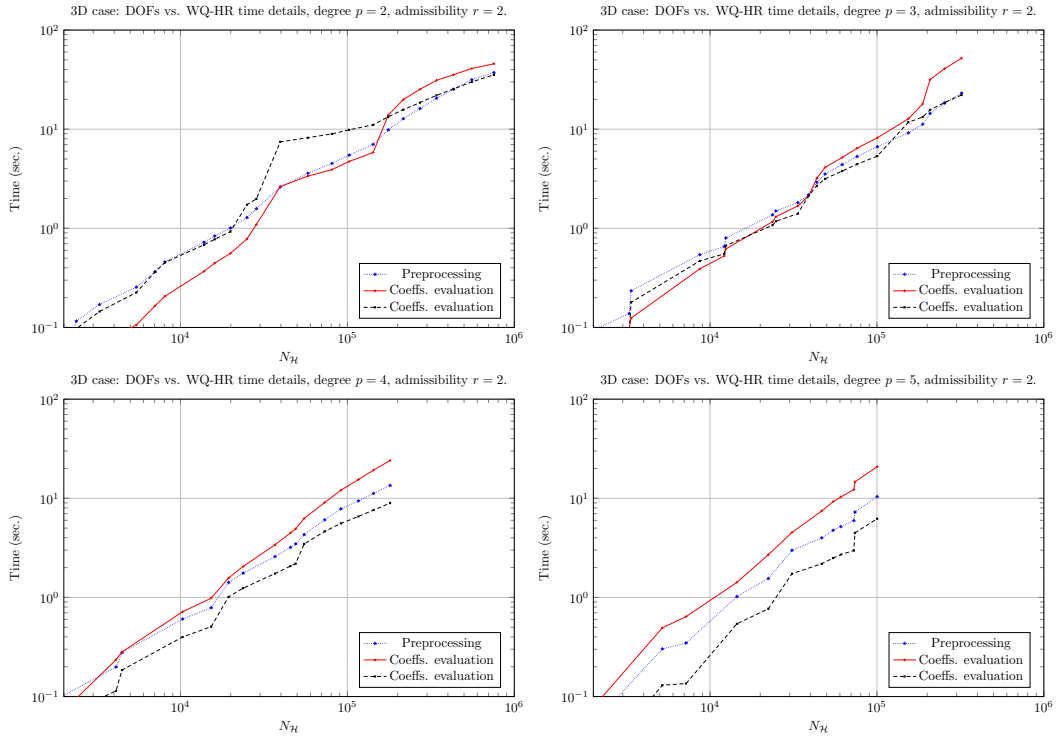


Figure 10: DOFs vs. the CPU time needed to run the WQ-HR algorithm: preprocessing (Algorithm 3) and matrix computation (Algorithm 5) for the 3D case and admissibility parameter $r = 2$. The cost for the matrix computation is split in the CPU cost for coefficient evaluations (Eq. (28), for $n = 0, \ldots, L$) and the CPU cost for executing the rest of the Algorithm 5.

Engineering 305 (2016) 217–240.

[25] 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, Computer Methods in Applied Mechanics and Engineering 316 (2017) 966–1004.

[26] D. Schillinger, S. J. Hossain, T. J. Hughes, Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 277 (2014) 1–45.

[27] C. Adam, T. J. Hughes, S. Bouabdallah, M. Zarroug, H. Maitournam, Selective and reduced numerical integrations for nurbs-based isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 284 (2015) 732–761.

[28] C. Bracco, C. Giannelli, R. Vázquez, Refinement algorithms for adaptive isogeometric methods with hierarchical splines, Axioms 7(3) (2018) 43.

[29] W. Dörfler, A convergent algorithm for poisson's equation, SIAM Jorunal of Nunerical Analysis 33 (1996) 1106–1124.

[30] M. S. Pauletti, M. Martinelli, N. Cavallini, P. Antolin Sanchez, Igatools: an isogeometric analysis library, SIAM Journal on Scientific Computing 37 (4) (2015) C465?C496.

[31] G. Sangalli, M. Tani, Matrix-free weighted quadrature for a computationally efficient isogeometric k-method, Computer Methods in Applied Mechanics and Engineering 338 (2018) 117–133.