Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 316 (2017) 606–622
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Fast formation of isogeometric Galerkin matrices by weighted quadrature

F. Calabrò[a], G. Sangalli[b,c,*], M. Tani[b]

[a] *DIEI, Università degli Studi di Cassino e del Lazio Meridionale, Italy*
[b] *Dipartimento di Matematica, Università degli Studi di Pavia, Italy*
[c] *Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes" del CNR, Pavia, Italy*

Available online 27 September 2016

Highlights

- We discuss the formation of isogeometric Galerkin matrices by row-loop and weighted quadrature.
- We give an estimate of the computational cost.
- We perform tests showing that the proposed algorithm over-performs other approaches.

Abstract

In this paper we propose an algorithm for the formation of matrices of isogeometric Galerkin methods. The algorithm is based on three ideas. The first is that we perform the external loop over the rows of the matrix. The second is that we calculate the row entries by weighted quadrature. The third is that we exploit the (local) tensor product structure of the basis functions. While all ingredients have a fundamental role for computational efficiency, the major conceptual change of paradigm with respect to the standard implementation is the idea of using weighted quadrature: the test function is incorporated in the integration weight while the trial function, the geometry parametrization and the PDEs coefficients form the integrand function. This approach is very effective in reducing the computational cost, while maintaining the optimal order of approximation of the method. Analysis of the cost is confirmed by numerical testing, where we show that, for p large enough, the time required by the floating point operations is less than the time spent in unavoidable memory operations (the sparse matrix allocation and memory write). The proposed algorithm allows significant time saving when assembling isogeometric Galerkin matrices for all the degrees of the test spline space and paves the way for a use of high-degree k-refinement in isogeometric analysis.

Keywords: Weighted quadrature; Isogeometric analysis; Splines; k-refinement

* Corresponding author at: Dipartimento di Matematica, Università degli Studi di Pavia, Italy.
E-mail addresses: calabro@unicas.it (F. Calabrò), giancarlo.sangalli@unipv.it (G. Sangalli), mattia.tani@unipv.it (M. Tani).

http://dx.doi.org/10.1016/j.cma.2016.09.013
0045-7825/

1. Introduction

Isogeometric analysis has been introduced by the seminal paper [1] as an extension of the classical finite element method. It is based on the idea of using the functions that are adopted for the geometry parametrization in computer aided design also to represent the numerical solution of the PDE of interest. These functions are splines, Non-Uniform Rational B-Splines (NURBS) and extensions. Many papers have demonstrated the advantage of isogeometric methods in various applications. For the interested reader, we refer to the book [2].

One interesting feature of isogeometric methods is the possibility of using high-degree high-regularity splines as they deliver higher accuracy per degree-of-freedom in comparison to C^0 finite elements [3–5]. However, the computational cost per degree-of-freedom is also higher for smooth splines, in currently available isogeometric codes. In practice, quadratic or cubic splines are preferred as they maximize computational efficiency.

The computational cost of a solver for a linear PDE problem is the sum of the cost of the formation of the system matrix and the cost of the solution of the linear system. The former is dominant in standard isogeometric codes already for low degree (see e.g. [6,7]). Recent papers in the literature have addressed this important issue (see e.g. [8,9]).

In this paper we adopt the following definition of *optimality*: an algorithm for the formation of the matrix of a Galerkin method is *optimal* if its computational cost is of the order of the number of non-zero entries of the matrix to be calculated. Optimal algorithms are known in the case of C^0 finite elements (see [10,11]). However, this is still an open problem for smooth splines.

We consider in this paper a d-dimensional scalar Poisson model problem on a single-patch domain, and an isogeometric tensor-product space of degree p and total dimension N_{DOF}, with $N_{\mathsf{DOF}} \gg p^d$. For the sake of simplicity, we focus on the case of C^{p-1} continuity, i.e., the typical setting of the so-called k-method (see e.g. [2]). The resulting stiffness matrix has $O(N_{\mathsf{DOF}}(2p+1)^d) \approx O(N_{\mathsf{DOF}} p^d)$ non-zero entries. Therefore, we assume $C N_{\mathsf{DOF}} p^d$ floating point[1] operations (FLOPs) is the (quasi)-optimal computational cost for the formation of the stiffness matrix. The algorithms currently used in isogeometric codes are suboptimal with respect to the degree p, that is, their cost grows with respect to the degree p faster than p^d.

The majority of isogeometric codes inherit a finite element architecture, which adopt an element-wise assembly loop with element-wise standard Gaussian quadrature (SGQ). Each local stiffness matrix has dimension $(p+1)^{2d}$ and each entry is calculated by quadrature on $(p+1)^d$ Gauss points. The total cost is $O(N_{\mathsf{EL}} p^{3d}) \approx O(N_{\mathsf{DOF}} p^{3d})$ FLOPs, where N_{EL} is the number of elements and, for the k-method, $N_{\mathsf{EL}} \approx N_{\mathsf{DOF}}$.

The standard way to reduce the cost is to reduce the number of quadrature points, for example by *reduced Gaussian* [12,13] (eventually corrected by variationally consistent constraints [14]) or *generalized Gaussian* quadrature (GGQ) [15–18]. To clarify GGQ, consider the mass matrix $\mathbb{M} = \{m_{i,j}\}$ whose entries, calculated on the parametric domain $\hat{\Omega} = [0,1]^d$, have the form

$$m_{i,j} = \int_{\hat{\Omega}} c(\boldsymbol{\zeta})\, \hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})\, \hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\, d\boldsymbol{\zeta}, \tag{1.1}$$

where $\hat{B}_{\boldsymbol{i}}$ and $\hat{B}_{\boldsymbol{j}}$ are the tensor-product B-spline, and c is a coefficient that incorporated the determinant Jacobian of the geometry mapping and other possible non-tensor product factors. The work [15] has explored the possibility of constructing and using GGQ quadrature of the kind

$$\int_{\hat{\Omega}} c(\boldsymbol{\zeta})\, \hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta}) \hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\, d\boldsymbol{\zeta} \approx \mathbb{Q}^{\mathrm{GGQ}}(c(\cdot)\, \hat{B}_{\boldsymbol{i}}(\cdot) \hat{B}_{\boldsymbol{j}}(\cdot)), \tag{1.2}$$

where the quadrature weights $w_{\boldsymbol{q}}^{\mathrm{GGQ}}$ and points $\boldsymbol{x}_{\boldsymbol{q}}^{\mathrm{GGQ}}$ of the quadrature rule $\mathbb{Q}^{\mathrm{GGQ}}(f(\cdot)) = \sum_{\boldsymbol{q}} w_{\boldsymbol{q}}^{\mathrm{GGQ}} f(\boldsymbol{x}_{\boldsymbol{q}}^{\mathrm{GGQ}})$ fulfil the exactness conditions

$$\int_{\hat{\Omega}} \hat{B}_{\boldsymbol{k}}^{2p}(\boldsymbol{\zeta}) d\boldsymbol{\zeta} = \mathbb{Q}^{\mathrm{GGQ}}(\hat{B}_{\boldsymbol{k}}^{2p}(\cdot)), \quad \forall \boldsymbol{k}. \tag{1.3}$$

Here $\{\hat{B}_{\boldsymbol{k}}^{2p}(\cdot)\}$ is the B-spline basis of degree $2p$ and continuity C^{p-1}. Exact integration of product of a pair of p degree splines $\hat{B}_{\boldsymbol{i}}(\cdot)\hat{B}_{\boldsymbol{j}}(\cdot)$ is then guaranteed by (1.3). Since the $w_{\boldsymbol{q}}^{\mathrm{GGQ}}$ and $\boldsymbol{x}_{\boldsymbol{q}}^{\mathrm{GGQ}}$ are not known analytically, they need

[1] Throughout the paper, C is a (reasonably small) constant that does not depend on N_{DOF} and p, and is in general different at each occurrence.

to be computed numerically as solution of the global non-linear problem (1.3), see [19–21], and the recent paper [22] where the problem is effectively solved by a Newton method with continuation. The paper [23] uses local exactness conditions instead of (1.3). The number of conditions in (1.3) is $\#\{\hat{B}_{\boldsymbol{k}}^{2p}(\cdot)\} \approx N_{\mathsf{DOF}}(p+1)^d \approx N_{\mathsf{EL}}(p+1)^d$, dropping the lower order terms, and therefore GGQ is expected to use about $N_{\mathsf{EL}}\left(\frac{p+1}{2}\right)^d$ quadrature points, with a saving of a factor 2^d with respect to SGQ.

The number of quadrature points is not the only issue to consider here, and indeed the element-wise assembling loop has a relevant role as well. On one hand, it allows the reuse of finite element available routines, which is a clear advantage as it greatly simplifies code development. On the other hand, it is intrinsically not optimal, as each elemental stiffness matrix has size $(p+1)^d$ and therefore the total computational cost is bounded from below by $CN_{\mathsf{EL}}p^{2d} \approx CN_{\mathsf{DOF}}p^{2d}$. This ideal threshold is approached by *sum-factorization*, that is, by arranging the computations in a way that exploits the tensor-product structure of multivariate spline, with a cost of $O(N_{\mathsf{DOF}}p^{2d+1})$ FLOPS, see [6].

Further cost reduction is possible with a change of paradigm from element-wise assembly. This has been explored in some recent papers. In [24] the integrand factor due to geometry and PDE coefficients is interpolated on the space of splines shape functions on a uniform knot vector, the same space where the approximation is considered, while the integrals arising are pre-computed in exact manner. The final cost of assembly in this case is $O(N_{\mathsf{DOF}}p^{2d})$. Another approach has been proposed in [25] where the stiffness matrix is approximated by a low-rank sum of R Kronecker matrices that can be efficiently formed thanks to their structure. This is a promising approach with computational cost of $O(N_{\mathsf{DOF}}Rp^d)$ FLOPs.

We propose in this paper a new algorithm which does not use the element-wise assembling loop. Instead, we loop over the matrix rows and we use a specifically designed *weighted quadrature* (WQ) rule for each row. In particular, the quadrature rule for the $\boldsymbol{i}$th row of $\mathbb{M}$ is as follows:

$$\int_{\hat{\Omega}} c(\boldsymbol{\zeta})\,\hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\,(\hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})d\boldsymbol{\zeta}) \approx \mathbb{Q}_{\boldsymbol{i}}^{\mathrm{WQ}}(c(\cdot)\,\hat{B}_{\boldsymbol{j}}(\cdot)), \quad \forall \boldsymbol{j}. \tag{1.4}$$

Unlike (1.2), in the right hand side of (1.4) the integrand function is $c(\cdot)\,\hat{B}_{\boldsymbol{j}}(\cdot)$ since the test function is incorporated into the integral weight (measure) $(\hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})d\boldsymbol{\zeta})$. The price to pay is that the quadrature weights depend on $\boldsymbol{i}$, while we select global quadrature points as suitable interpolation points that do not depend on $\boldsymbol{i}$. Again, the quadrature weights are not known analytically and need to be computed numerically as solution of the exactness conditions

$$\int_{\hat{\Omega}} \hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\,(\hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})d\boldsymbol{\zeta}) = \mathbb{Q}_{\boldsymbol{i}}^{\mathrm{WQ}}(\hat{B}_{\boldsymbol{j}}). \tag{1.5}$$

However, the exactness conditions (1.5) are linear with respect to the weights. Furthermore, (1.5) is a local problem as the weights outside $\mathrm{supp}(\hat{B}_{\boldsymbol{i}})$ can be set to zero a priori. The knot vectors do not need to be uniform with this approach.

The number of exactness conditions of (1.5) is $\#\{\hat{B}_{\boldsymbol{j}}(\cdot)\} = N_{\mathsf{DOF}}$. This is lower than the number of conditions of (1.3), which is $\#\{\hat{B}_{\boldsymbol{k}}^{2p}(\cdot)\} \approx N_{\mathsf{DOF}}(p+1)^d$. Hence, the main advantage of the WQ with respect to GGQ is that the former requires significantly fewer quadrature points. In the case of maximum regularity only 2 points are needed in each direction sufficiently far away from the boundary, while $p+1$ points are taken on boundary knot-spans along directions that end on the boundary. Adopting sum-factorization (see [6]), the proposed algorithm has a total computational cost of $O(N_{\mathsf{DOF}}p^{d+1})$ FLOPs.

In our numerical benchmarking, performed in MATLAB, we have compared the standard GeoPDEs 3.0 (see [26, 27]) mass matrix formation, based on element-loop SGQ, with our row-loop WQ-based algorithm, showing the impressive advantage. WQ speedup is more than a factor of four for quadratics and rapidly grows with the degree p. For example, the mass matrix on a 20^3 elements grid is calculated in about 62 h vs 27 s for degree $p = 10$. For high p, the asymptotic growth of the computational time is slower than the estimated cost from FLOPs counting: for all degrees of practical interest the growth we have measured is at most $CN_{\mathsf{DOF}}p^d$, that is, optimal. This is due to the fact that the memory operations dominate: in particular we have verified that the matrix formation time with our implementation is mainly used in allocation (MATLAB's `sparse` call) and memory write at least for sufficiently high degree p.

The WQ we propose is designed in order to fit into the mathematical theory that guarantees optimal order of convergence of the method. This theory is based on the Strang lemma [28,29]. We do not enter into this topic, which is technical, and postpone it to a further work. In this paper we give numerical evidence of optimal convergence on a simple 1D benchmark.

We also do not investigate parallelization in this paper, however we think the proposed algorithm is well suited for a parallel implementation since each matrix row is calculated independently, which should alleviate the race condition of typical finite element-wise assembly (see [30] for details).

The outline of the paper is as follows. In Section 2, we present the idea of the WQ rules for univariate B-splines. In Section 3, we briefly discuss the use of isogeometric analysis on a model problem, and fix the notation for the following sections. In Section 4, we extend the construction of WQ rules to the multivariate case; a pseudo-code is presented in Section 5, where the computational cost is also discussed. In Section 6, we give details on the application of the WQ rules for the formation of the mass matrix. In Section 7 we test the procedure: a simple 1D test is performed in order to confirm accuracy and tests are presented in order to compare time needed for the formation of mass matrices in 3D. A complete benchmarking of the proposed procedure is beyond the scope of the present paper and will be the subject of a forthcoming paper. Finally, in Section 8 we draw conclusions.

2. Weighted quadrature

Assume we want to compute integrals of the kind:

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_j(\zeta)\,d\zeta, \tag{2.1}$$

where $\{\hat{B}_i\}_{i=1,\dots,n_{\mathsf{DOF}}}$ are p-degree univariate B-spline basis functions defined on the parametric patch $[0,1]$. We denote by $\boldsymbol{\chi}$ the knot vector of distinct knots that define the univariate B-splines $\hat{B}_i(\zeta)$. Moreover we define *knot-spans* as the intervals $[\chi_k, \chi_{k+1}]$, $k = 1,\dots,n_{\mathsf{EL}}$, where $n_{\mathsf{EL}} := (\#\boldsymbol{\chi}) - 1$. The knot vector

$$\Xi := \left\{\xi_1, \xi_2, \dots, \xi_{n_{\mathsf{KNT}}}\right\} \tag{2.2}$$

that defines the univariate B-splines contains knots with repetitions depending on the regularity: if a knot χ_k has multiplicity $p-r$ then the univariate spline is C^r continuous at χ_k. For simplicity, we consider $r = p-1$ throughout this paper. Though it is not difficult to consider arbitrary r, the proposed strategy takes advantage of high regularity. In order to focus on the relevant properties, we restrict our attention in this section to the uniform knot-spans, i.e. $\chi_{k+1} - \chi_k = h \forall k = 1,\dots,n_{\mathsf{EL}} - 1$. Moreover, we do not consider boundary functions, so we assume that the knot vector is periodic. Being in the context of Galerkin method, $\hat{B}_i(\zeta)$ is denoted as a test function and $\hat{B}_j(\zeta)$ as a trial function.

We are interested in a fixed point quadrature rule. In the lowest degree case, $p = 1$, exact integration is performed by a composite Cavalieri–Simpson rule (note that in this case this quadrature is also the Gauss–Lobatto 3 points rule):

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_j(\zeta)\,d\zeta = \mathbb{Q}^{CS}(\hat{B}_i\,\hat{B}_j) = \sum_q w_q^{CS}\hat{B}_i(x_q^{CS})\,\hat{B}_j(x_q^{CS}), \tag{2.3}$$

where x_q^{CS} are the quadrature points and w_q^{CS} the relative weights, see Fig. 1. In the above hypotheses the points x_q^{CS} are the knots and the midpoints of the knot-spans and $w_q^{CS} = \frac{h}{3}$ on knots and $w_q^{CS} = \frac{2h}{3}$ on midpoints.

Unbalancing the role of the test and the trial factors in (2.3), we can see it as a weighted quadrature:

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_j(\zeta)\,d\zeta = \mathbb{Q}_i^{WQ}(\hat{B}_j) = \sum_q w_{q,i}^{WQ}\hat{B}_j(x_{q,i}^{WQ}), \tag{2.4}$$

where $x_q^{CS} = x_{q,i}^{WQ}$ and $w_{q,i}^{WQ} = \hat{B}_i(x_{q,i}^{WQ})w_q^{CS}$. Because of the local support of the function $\hat{B}_i$ only in three points the quadrature $\mathbb{Q}_i^{WQ}$ is non-zero and the weights are equal to $\frac{h}{3}$ see Fig. 1.

If we go to higher degree, we need more quadrature points in (2.3). For p-degree splines the integrand $\hat{B}_i\hat{B}_j$ is a piecewise polynomial of degree $2p$ and an element-wise integration requires $2p+1$ equispaced points, or $p+1$

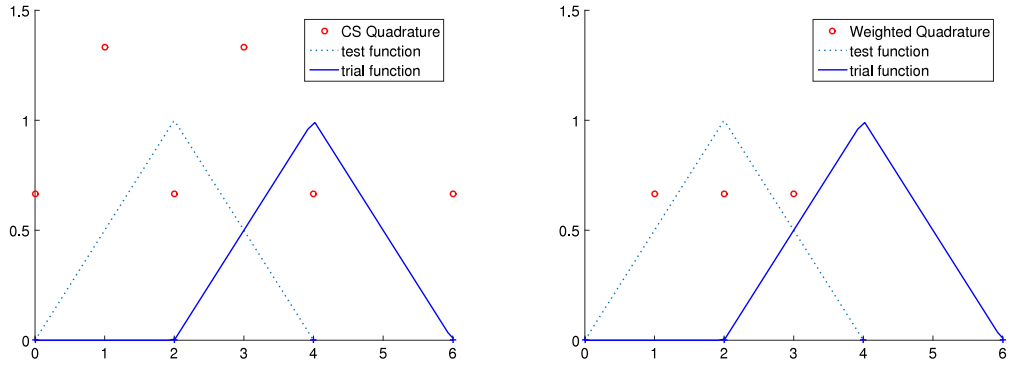

Fig. 1. Quadrature rule for B-spline with $p = 1$. Cavalieri–Simpson quadrature rule (on the left) and its interpretation as weighted quadrature (on the right). The active points and weights for $\mathbb{Q}_i^{WQ}$ are highlighted. In this and the next figure we set $h = 2$ so that the quadrature points coincide with the integers, being the knots and the midpoints of the knot-spans.

Gauss points, or about $p/2$ points with generalized Gaussian integration (see [15,23,16,31]). On the other hand, we can generalize (2.4) to higher degree still using as quadrature points only the knots and midpoints of the knot spans. Indeed this choice ensures that, for each basis function $\hat{B}_i$, $i = 1, \dots, n_{\mathsf{DOF}}$, there are $2p + 1$ "active" quadrature points where $\hat{B}_i$ is nonzero. Therefore we can compute the $2p + 1$ quadrature weights by imposing conditions for the $2p + 1$ B-splines $\hat{B}_j$ that need to be exactly integrated. Clearly, the advantage of the weighted quadrature approach is that its computational complexity, i.e., the total number of quadrature points, is independent of p.

For the sake of clarity, we first consider the case $p = 1$ in detail. The exactness conditions are:

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i-1}(\zeta)\,d\zeta = \frac{h}{6} = \mathbb{Q}_i^{WQ}(\hat{B}_{i-1}) = \frac{1}{2}w_{1,i}^{WQ}, \tag{2.5}$$

$$\int_0^1 \hat{B}_i^2(\zeta)\,d\zeta = \frac{2h}{3} = \mathbb{Q}_i^{WQ}(\hat{B}_i) = \frac{1}{2}w_{1,i}^{WQ} + w_{2,i}^{WQ} + \frac{1}{2}w_{3,i}^{WQ}, \tag{2.6}$$

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i+1}(\zeta)\,d\zeta = \frac{h}{6} = \mathbb{Q}_i^{WQ}(\hat{B}_{i+1}) = \frac{1}{2}w_{3,i}^{WQ}. \tag{2.7}$$

Then it is easy to compute $w_{q,i}^{WQ} = h/3,\ \forall q = 1, 2, 3$.

In the case $p = 2$, five points are active and we have five exactness equations:

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i-2}(\zeta)\,d\zeta = \frac{h}{120} = \mathbb{Q}_i^{WQ}(\hat{B}_{i-2}) = \frac{1}{8}w_{1,i}^{WQ}, \tag{2.8}$$

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i-1}(\zeta)\,d\zeta = \frac{26h}{120} = \mathbb{Q}_i^{WQ}(\hat{B}_{i-1}) = \frac{3}{4}w_{1,i}^{WQ} + \frac{1}{2}w_{2,i}^{WQ} + \frac{1}{8}w_{3,i}^{WQ}, \tag{2.9}$$

$$\int_0^1 \hat{B}_i^2(\zeta)\,d\zeta = \frac{66h}{120} = \mathbb{Q}_i^{WQ}(\hat{B}_i) = \frac{1}{8}w_{1,i}^{WQ} + \frac{1}{2}w_{2,i}^{WQ} + \frac{3}{4}w_{3,i}^{WQ} + \frac{1}{2}w_{4,i}^{WQ} + \frac{1}{8}w_{5,i}^{WQ}, \tag{2.10}$$

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i+1}(\zeta)\,d\zeta = \frac{26h}{120} = \mathbb{Q}_i^{WQ}(\hat{B}_{i+1}) = \frac{1}{8}w_{3,i}^{WQ} + \frac{1}{2}w_{4,i}^{WQ} + \frac{3}{4}w_{5,i}^{WQ}, \tag{2.11}$$

$$\int_0^1 \hat{B}_i(\zeta)\,\hat{B}_{i+2}(\zeta)\,d\zeta = \frac{h}{120} = \mathbb{Q}_i^{WQ}(\hat{B}_{i+2}) = \frac{1}{8}w_{5,i}^{WQ}. \tag{2.12}$$

In the previous calculation we have used the usual properties of B-splines that can be found, e.g., in [32, Section 4.4]. The system can be solved and leads to the following solution $w_{q,i}^{WQ} = \frac{h}{30}\,[2, 7, 12, 7, 2]$.

When $p = 3$, the same approach gives $w_{q,i}^{WQ} = h\left[\frac{1}{105}, \frac{3}{35}, \frac{5}{21}, \frac{1}{3}, \frac{5}{21}, \frac{3}{35}, \frac{1}{105}\right]$. These computations are reported in Fig. 2.

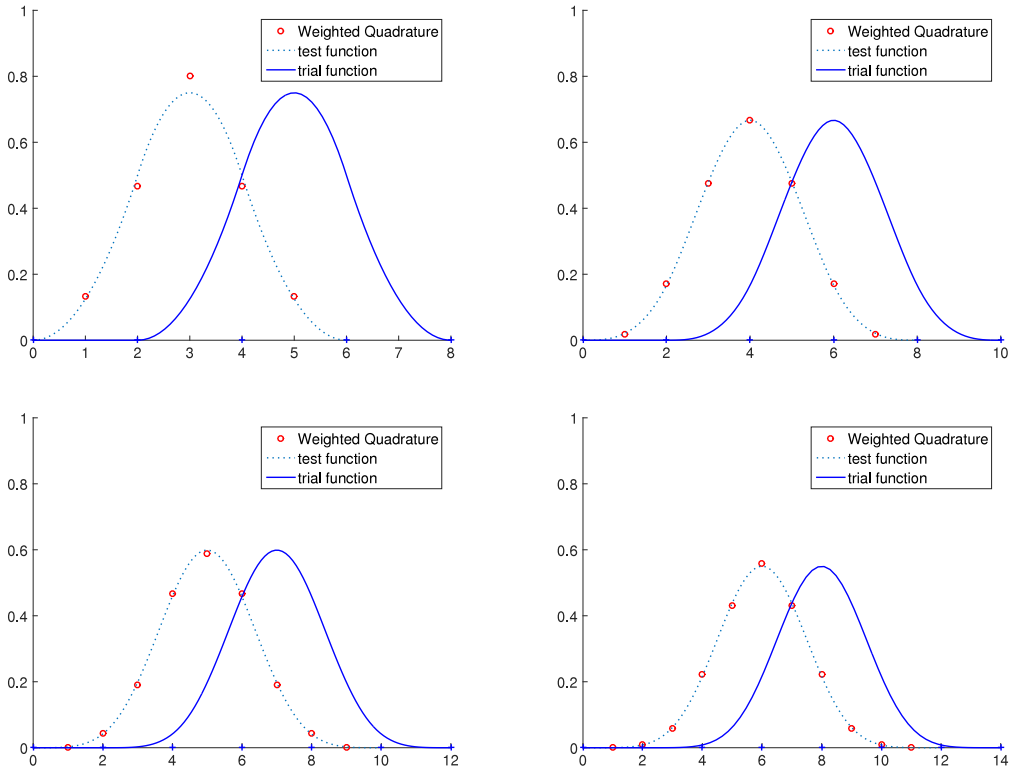

Fig. 2. Weighted quadrature rule for B-spline with various degrees, namely $p = 2$ (upper left), $p = 3$ (upper right), $p = 4$ (lower left) and $p = 5$ (lower right). Interestingly, we see that the weights displace around the values of the basis function (up to the scale factor $h/2$, which in this case is simply 1). In particular, this gives numerical evidence of the positivity of the weights, which in turn implies the stability of the rules.

In general case (arbitrary degree and non-uniform spacing, boundary functions, lower regularity...) the rule can be computed numerically as solution of a linear system, see Section 5.

Given a weighted quadrature rule of the kind above, we are then interested in using it for the approximate calculation of integrals as:

$$\int_0^1 c(\zeta)\hat{B}_i(\zeta)\,\hat{B}_j(\zeta)\,d\zeta \approx \mathbb{Q}_i^{WQ}\left(c(\cdot)\hat{B}_j(\cdot)\right) = \sum_q w_{q,i}^{WQ} c(x_{q,i}^{WQ})\hat{B}_j(x_{q,i}^{WQ}). \tag{2.13}$$

For a non-constant function $c(\cdot)$, (2.13) is in general just an approximation. In particular, the symmetry of the integral is not preserved, that is $\mathbb{Q}_i^{WQ}\left(c(\cdot)\hat{B}_j(\cdot)\right)$ is different from $\mathbb{Q}_j^{WQ}\left(c(\cdot)\hat{B}_i(\cdot)\right)$. Consider, for example, the case $p = 3$ derived above and apply the weighted quadrature rules to the linear function $c(\zeta) = \zeta$ in the case $j = i + 1$. For simplicity we take $h = 2$ so that the quadrature points are $x_{q,i}^{WQ} = [1 : 7]$. Then:

$$\mathbb{Q}_i^{WQ}\left(c(\cdot)\hat{B}_j(\cdot)\right) = \frac{10}{21}3\frac{1}{48} + \frac{2}{3}4\frac{1}{6} + \frac{10}{21}5\frac{23}{48} + \frac{6}{35}6\frac{2}{3} + \frac{2}{105}7\frac{23}{48} \approx 2.3647,$$
$$\mathbb{Q}_j^{WQ}\left(c(\cdot)\hat{B}_i(\cdot)\right) = \frac{2}{105}3\frac{23}{48} + \frac{6}{35}4\frac{2}{3} + \frac{10}{21}5\frac{23}{48} + \frac{2}{3}6\frac{1}{6} + \frac{10}{21}7\frac{1}{48} \approx 2.3615.$$

A detailed mathematical analysis of the quadrature error of weighted quadrature is of key interest, especially in the context of isogeometric Galerkin methods. This is however beyond the scope of this paper and for its importance deserves future work.

3. Integral arising in isogeometric Galerkin methods

We consider the model reaction–diffusion problem

$$\begin{cases} -\nabla^2 u + u = f & \text{on} \quad \Omega, \\ u = 0 & \text{on} \quad \partial\Omega. \end{cases} \tag{3.1}$$

Its Galerkin approximation on a discrete space V requires the formation of the stiffness matrix S and mass matrix M whose entries are

$$s_{\boldsymbol{i},\boldsymbol{j}} = \int_{\Omega} \nabla R_{\boldsymbol{i}}(\boldsymbol{x}) \nabla R_{\boldsymbol{j}}(\boldsymbol{x}) d\boldsymbol{x}, \tag{3.2}$$

$$m_{\boldsymbol{i},\boldsymbol{j}} = \int_{\Omega} R_{\boldsymbol{i}}(\boldsymbol{x}) R_{\boldsymbol{j}}(\boldsymbol{x}) d\boldsymbol{x}. \tag{3.3}$$

$R_{\boldsymbol{i}}$ and $R_{\boldsymbol{j}}$ being two basis functions in V. The dimension of the space V is $N_{\mathsf{DOF}} := \#V$. Non-constant coefficients could be included as well.

In the isogeometric framework, Ω is given by a spline or NURBS parametrization. Our notation follows [5, Section 4] and [6]. For the sake of simplicity, we assume Ω is given by a d-dimensional single patch spline representation, then it is of the form:

$$\Omega = \boldsymbol{F}(\hat{\Omega}), \text{ with } \boldsymbol{F}(\boldsymbol{\zeta}) = \sum_{\boldsymbol{i}} \boldsymbol{C}_{\boldsymbol{i}} \hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta}),$$

where $\boldsymbol{C}_{\boldsymbol{i}}$ are the control points and $\hat{B}_{\boldsymbol{i}}$

We denote by $\boldsymbol{\chi}_l$ the knot vector of distinct knots that define the univariate B-splines $\hat{B}_{i_l}(\zeta_l)$ along the lth direction. For each direction we have *knot-spans* as the intervals $[\chi_{l,k}, \chi_{l,k+1}]$, $k = 1, \ldots, n_{\mathsf{EL},l}$, where $n_{\mathsf{EL},l} := (\#\boldsymbol{\chi}_l) - 1$. By cartesian product, they form a mesh of $N_{\mathsf{EL}} = \prod_{l=1}^{d} n_{\mathsf{EL},l}$ *elements* on $\hat{\Omega}$. The knot vector, with possibly repeated knots, that defines the univariate B-spline space in the lth direction is denoted as

$$\Xi_l := \left\{ \xi_1, \xi_2, \ldots, \xi_{n_{\mathsf{KNT},l}} \right\}. \tag{3.4}$$

As in Section 2, we restrict to the case of maximum regularity and allow repeated knots only at the endpoints of the open knot vector.

The number of knots in each knot vector $n_{\mathsf{KNT},l} := \#\Xi_l$ is related to the number of degrees of freedom by $n_{\mathsf{DOF},l} + (p+1) = n_{\mathsf{KNT},l}$. No assumptions are made on the length of the elements. In our FLOPs counts, we always assume $p \ll n_{\mathsf{DOF},l}$, and then $n_{\mathsf{DOF},l} \approx n_{\mathsf{KNT},l} \approx n_{\mathsf{EL},l}$.

The multivariate B-splines are tensor-product of univariate B-splines:

$$\hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta}) = \hat{B}_{i_1}(\zeta_1) \ldots \hat{B}_{i_d}(\zeta_d). \tag{3.5}$$

Above, $\boldsymbol{i} = (i_1, \ldots, i_d)$ is a multi-index that, with abuse of notation, is occasionally as a scalar index, as in $\boldsymbol{i} = 1, \ldots, N_{\mathsf{DOF}}$, with the relation $\boldsymbol{i} \equiv 1 + \sum_{l=1}^{d} (n_{\mathsf{DOF},1} \ldots n_{\mathsf{DOF},l-1})(i_l - 1)$. We have $N_{\mathsf{DOF}} = \prod_{l=1}^{d} n_{\mathsf{DOF},l}$.

Based on the isogeometric/isoparametric paradigm, the basis functions $R_{\boldsymbol{i}}$ used in (3.2)–(3.3) are defined as $R_{\boldsymbol{i}} = \hat{B}_{\boldsymbol{i}} \circ \boldsymbol{F}^{-1}$; integrals are computed by change of variable. Summarizing, we are interested in the computation of (3.3) after change of variable, $\mathbb{M} = \{m_{\boldsymbol{i},\boldsymbol{j}}\} \in \mathbb{R}^{N_{\mathsf{DOF}} \times N_{\mathsf{DOF}}}$ where:

$$m_{\boldsymbol{i},\boldsymbol{j}} = \int_{\hat{\Omega}} \hat{B}_{\boldsymbol{i}}\, \hat{B}_{\boldsymbol{j}}\, \det \hat{\boldsymbol{D}}\boldsymbol{F}\, d\boldsymbol{\zeta}.$$

For notational convenience we write:

$$m_{\boldsymbol{i},\boldsymbol{j}} = \int_{\hat{\Omega}} \hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})\, \hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\, c(\boldsymbol{\zeta})\, d\boldsymbol{\zeta}. \tag{3.6}$$

In more general cases, the factor c incorporates the coefficient of the equation and, for NURBS functions, the polynomial denominator. Similarly for the stiffness matrix $\mathbb{S} = \{s_{\boldsymbol{i},\boldsymbol{j}}\} \in \mathbb{R}^{N_{\mathsf{DOF}} \times N_{\mathsf{DOF}}}$ we have:

$$\begin{aligned} s_{\boldsymbol{i},\boldsymbol{j}} &= \int_{\hat{\Omega}} \left(\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}\hat{\nabla}\hat{B}_{\boldsymbol{i}}\right)^T \left(\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}\hat{\nabla}\hat{B}_{\boldsymbol{j}}\right) \det\hat{\boldsymbol{D}}\boldsymbol{F}\, d\boldsymbol{\zeta} \\ &= \int_{\hat{\Omega}} \hat{\nabla}\hat{B}_{\boldsymbol{i}}^T \left(\left[\hat{\boldsymbol{D}}\boldsymbol{F}^{-1}\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}\right] \det\hat{\boldsymbol{D}}\boldsymbol{F}\right) \hat{\nabla}\hat{B}_{\boldsymbol{j}}\, d\boldsymbol{\zeta} \end{aligned}$$

which we write in compact form:

$$s_{\boldsymbol{i},\boldsymbol{j}} = \sum_{l,m=1}^{d} \int_{\hat{\Omega}} \left(\hat{\nabla}\hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})\right)_l c_{l,m}(\boldsymbol{\zeta}) \left(\hat{\nabla}\hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})\right)_m d\boldsymbol{\zeta}. \tag{3.7}$$

Here we have denoted by $\left\{c_{l,m}(\boldsymbol{\zeta})\right\}_{l,m=1,\dots,d}$ the following matrix:

$$c_{l,m}(\boldsymbol{\zeta}) = \left\{\left[\hat{\boldsymbol{D}}\boldsymbol{F}^{-1}(\boldsymbol{\zeta})\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}(\boldsymbol{\zeta})\right] \det\hat{\boldsymbol{D}}\boldsymbol{F}(\boldsymbol{\zeta})\right\}_{l,m}. \tag{3.8}$$

The number of non-zero elements N_{NZ} of $\mathbb{M}$ and $\mathbb{S}$ (the same for simplicity) depends on the polynomial degree p and the required regularity r. We introduce the following sets, where the support is considered an open set:

$$\mathcal{K}_{l,i_l} = \left\{k \in \{1,\dots,n_{\mathsf{EL},l}\}\, s.t.\,]\chi_{k-1},\, \chi_k[\subset \operatorname{supp}\left(\hat{B}_{i_l}\right)\right\}, \tag{3.9}$$

$$\mathcal{I}_{l,i_l} = \left\{j_l \in \{1,\dots,n_{\mathsf{DOF},l}\}\, s.t.\, \hat{B}_{i_l} \cdot \hat{B}_{j_l} \neq 0\right\}; \tag{3.10}$$

and the related multi-indices as:

$$\mathcal{K}_{\boldsymbol{i}} = \prod_{l=1}^{d} \mathcal{K}_{l,i_l} \qquad \mathcal{I}_{\boldsymbol{i}} = \prod_{l=1}^{d} \mathcal{I}_{l,i_l}. \tag{3.11}$$

We have $\#\mathcal{I}_{l,i} \leq (2p+1)$ and $N_{\mathsf{NZ}} = O(N_{\mathsf{DOF}}\, p^d)$. In particular, with maximal regularity in the case $d = 1$ one has $N_{\mathsf{NZ}} = (2p+1)N_{\mathsf{DOF}} - p(p+1)$.

4. Computation of the WQ rules

Consider the calculation of the mass matrix (3.3). The first step is to write the integral in a nested way, as done in [6]:

$$\begin{aligned} m_{\boldsymbol{i},\boldsymbol{j}} &= \int_{\hat{\Omega}} \hat{B}_{\boldsymbol{i}}(\boldsymbol{\zeta})\hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta})c(\boldsymbol{\zeta})\, d\boldsymbol{\zeta} \\ &= \int_0^1 \hat{B}_{i_1}(\zeta_1)\hat{B}_{j_1}(\zeta_1) \left[\int_0^1 \hat{B}_{i_2}(\zeta_2)\hat{B}_{j_2}(\zeta_2) \cdots \left[\int_0^1 \hat{B}_{i_d}(\zeta_d)\hat{B}_{j_d}(\zeta_d)c(\boldsymbol{\zeta})\, d\zeta_d\right] \cdots d\zeta_2\right] d\zeta_1. \end{aligned}$$

Our idea is to isolate the *test function* $\hat{B}_{i_l}$ univariate factors in each univariate integral and to consider it as a weight for the construction of the weighted quadrature (WQ) rule. This leads to a quadrature rule for each i_l that is:

$$\begin{aligned} m_{\boldsymbol{i},\boldsymbol{j}} \approx \tilde{m}_{\boldsymbol{i},\boldsymbol{j}} &= \mathbb{Q}_{\boldsymbol{i}}^{WQ}\left(\hat{B}_{\boldsymbol{j}}(\zeta)c(\boldsymbol{\zeta})\right) = \mathbb{Q}_{\boldsymbol{i}}\left(\hat{B}_{\boldsymbol{j}}(\zeta)c(\boldsymbol{\zeta})\right) \\ &= \mathbb{Q}_{i_1}\left(\hat{B}_{j_1}(\zeta_1)\mathbb{Q}_{i_2}\left(\cdots \mathbb{Q}_{i_d}\left(\hat{B}_{j_d}(\zeta_d)c(\boldsymbol{\zeta})\right)\right)\right). \end{aligned} \tag{4.1}$$

Notice that we drop from now on the label WQ used in the introduction in order to simplify notation. The key ingredients for the construction of the quadrature rules that preserve the optimal approximation properties are the exactness requirements. Roughly speaking, exactness means that in (4.1) we have $m_{\boldsymbol{i},\boldsymbol{j}} = \tilde{m}_{\boldsymbol{i},\boldsymbol{j}}$ whenever c is a constant coefficient. When the stiffness term is considered, also terms with derivatives have to be considered.

We introduce the notation:

$$\begin{aligned}
\mathbb{I}^{(0,0)}_{l,i_l,j_l} &:= \int_0^1 \hat{B}_{i_l}(\zeta_l)\hat{B}_{j_l}(\zeta_l)\,d\zeta_l \\
\mathbb{I}^{(1,0)}_{l,i_l,j_l} &:= \int_0^1 \hat{B}'_{i_l}(\zeta_l)\hat{B}_{j_l}(\zeta_l)\,d\zeta_l \\
\mathbb{I}^{(0,1)}_{l,i_l,j_l} &:= \int_0^1 \hat{B}_{i_l}(\zeta_l)\hat{B}'_{j_l}(\zeta_l)\,d\zeta_l \\
\mathbb{I}^{(1,1)}_{l,i_l,j_l} &:= \int_0^1 \hat{B}'_{i_l}(\zeta_l)\hat{B}'_{j_l}(\zeta_l)\,d\zeta_l .
\end{aligned} \tag{4.2}$$

For each integral in (4.2) we define a quadrature rule: we look for

- points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}} = (\tilde{x}_{1,q_1}, \tilde{x}_{2,q_2}, \ldots, \tilde{x}_{d,q_d})$ with $q_l = 1, \ldots, n_{\mathsf{QP},l}$, with N_{QP} is $\#\{\tilde{\boldsymbol{x}}\} = \prod_{l=1}^{d} n_{\mathsf{QP},l}$;
- for each index $i_l = 1, \ldots, n_{\mathsf{DOF},l}$; $l = 1, \ldots, d$, four quadrature rules such that:

$$\begin{aligned}
\mathbb{Q}^{(0,0)}_{i_l}(f) &:= \sum_{q_l=1}^{n_{\mathsf{QP},l}} w^{(0,0)}_{l,i_l,q_l} f(\tilde{x}_{l,q_l}) \approx \int_0^1 f(\zeta_l)\hat{B}_{i_l}(\zeta_l)d\zeta_l \,; \\
\mathbb{Q}^{(1,0)}_{i_l}(f) &:= \sum_{q_l=1}^{n_{\mathsf{QP},l}} w^{(1,0)}_{l,i_l,q_l} f(\tilde{x}_{l,q_l}) \approx \int_0^1 f(\zeta_l)\hat{B}_{i_l}(\zeta_l)d\zeta_l \,; \\
\mathbb{Q}^{(0,1)}_{i_l}(f) &:= \sum_{q_l=1}^{n_{\mathsf{QP},l}} w^{(0,1)}_{l,i_l,q_l} f(\tilde{x}_{l,q_l}) \approx \int_0^1 f(\zeta_l)\hat{B}'_{i_l}(\zeta_l)d\zeta_l \,; \\
\mathbb{Q}^{(1,1)}_{i_l}(f) &:= \sum_{q_l=1}^{n_{\mathsf{QP},l}} w^{(1,1)}_{l,i_l,q_l} f(\tilde{x}_{l,q_l}) \approx \int_0^1 f(\zeta_l)\hat{B}'_{i_l}(\zeta_l)d\zeta_l
\end{aligned} \tag{4.3}$$

fulfilling the exactness requirement:

$$\begin{aligned}
\mathbb{Q}^{(0,0)}_{i_l}(\hat{B}_{j_l}) &= \mathbb{I}^{(0,0)}_{l,i_l,j_l} \\
\mathbb{Q}^{(1,0)}_{i_l}(\hat{B}'_{j_l}) &= \mathbb{I}^{(1,0)}_{l,i_l,j_l} \\
\mathbb{Q}^{(0,1)}_{i_l}(\hat{B}_{j_l}) &= \mathbb{I}^{(0,1)}_{l,i_l,j_l} \\
\mathbb{Q}^{(1,1)}_{i_l}(\hat{B}'_{j_l}) &= \mathbb{I}^{(1,1)}_{l,i_l,j_l}
\end{aligned} \quad , \ \forall j_l \in \mathcal{I}_{l,i_l} \, . \tag{4.4}$$

For stability we also require that the quadrature rules $\mathbb{Q}^{(\cdot,\cdot)}_{i_l}$ have support included in the support of $\hat{B}_{i_l}$, that is

$$q_l \notin \mathcal{Q}_{l,i_l} \Rightarrow w^{(\cdot,\cdot)}_{l,i_l,q_l} = 0 \tag{4.5}$$

where $\mathcal{Q}_{l,i_l} := \left\{ q_l \in 1, \ldots, n_{\mathsf{QP},l} \text{ s.t. } \tilde{x}_{l,q_l} \in \operatorname{supp}\left(\hat{B}_{i_l}\right) \right\}$; recall that here the support of a function is considered an open set. Correspondingly, we introduce the set of multi-indices $\mathcal{Q}_{\boldsymbol{i}} := \prod_{l=1}^{d} \mathcal{Q}_{l,i_l}$.

Once the points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}}$ are fixed, the quadrature rules have to be determined by the exactness requirements, that are a system of linear equations of the unknown weights (each of (4.4)). For that we require

$$\#\mathcal{Q}_{l,i_l} \geq \#\mathcal{I}_{l,i_l} . \tag{4.6}$$

See Remark 4.2 for a discussion on the well-posedness of the linear systems for the weights.

Remark 4.1 (*The Choice of Quadrature Points*)**.** The construction of a global grid of quadrature points is done in order to save computations. For the case of maximum C^{p-1} regularity considered here, our choice for quadrature points is endpoints (knots) and midpoints of all internal knot-spans, while for the boundary knot-spans (i.e. those that are adjacent to the boundary of the parameter domain $\hat{\Omega}$) we take $p+1$ equally spaced points. Globally $N_{\mathsf{QP}} \approx 2^d N_{\mathsf{EL}} = O(N_{\mathsf{DOF}})$ considering only the dominant term (remember that $n_{\mathsf{EL},l} \gg p$). In Fig. 3 we plot the

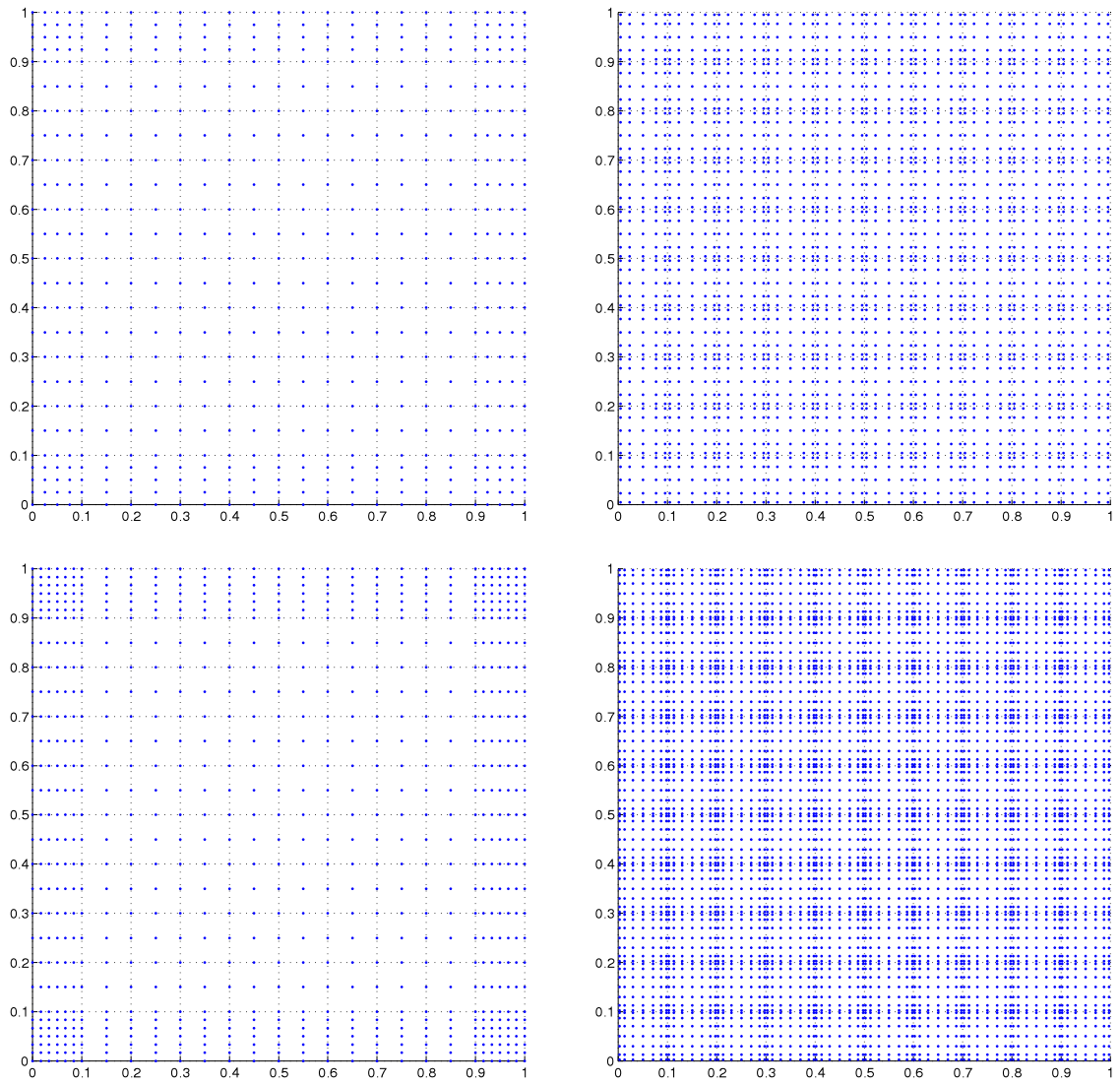

Fig. 3. Comparison between quadrature rules. On the first line, the quadrature points needed for the case $p = 4$, $n_{\mathsf{EL}} = 10$, $d = 2$, in the second line the case $p = 6$. On the left panel the proposed WQ rule, on the right the SGQ rule.

quadrature points grid, and a comparison is made with respect to element-by-element standard Gaussian quadrature (SGQ) points.

Remark 4.2 (*Computation of Quadrature Weights*)**.** Given the quadrature points, the quadrature weights are selected in order to fulfil (4.4)–(4.5). When $\#\mathcal{Q}_{l,i_l} > \#\mathcal{I}_{l,i_l}$ the quadrature weights are not uniquely given from (4.4) to (4.5) and are selected by a minimum norm condition. In all cases with our choice of quadrature points and thanks to the Schoenberg–Whitney interpolation theorem [33, Theorem 2 ¶XIII] we can guarantee that the quadrature weights fulfilling the above conditions exist.

Remark 4.3 (*Alternative Choices of the Quadrature Points*)**.** If there is no need for a global grid of quadrature points (e.g., the cost of calculation of the coefficients is negligible), it is possible to have quadrature points that depend on the index i_l, as for the weights. Then, one can construct weighted Gaussian quadrature rules (see e.g. [31]) in order to minimize the number of quadrature points associated to each row of the stiffness matrix.

5. Pseudo-codes and computational cost

In order to simplify the FLOPs count, we assume $n_{\mathsf{EL}} := n_{\mathsf{EL},1} = n_{\mathsf{EL},2} = \cdots = n_{\mathsf{EL},d}$ and $n_{\mathsf{DOF}} := n_{\mathsf{DOF},1} = n_{\mathsf{DOF},2} = \cdots = n_{\mathsf{DOF},d}$. We then have $N_{\mathsf{EL}} = n_{\mathsf{EL}}^d$, $N_{\mathsf{DOF}} = n_{\mathsf{DOF}}^d$, etc. We consider the case of maximum regularity $r = p - 1$ and $n_{\mathsf{EL}} \gg p$ that implies $n_{\mathsf{EL}} \approx n_{\mathsf{DOF}}$. We recall that:

$$\begin{aligned} \#\mathcal{K}_{l,i_l} &\leq p+1 \\ \#\mathcal{I}_{l,i_l} &\leq 2p+1. \end{aligned} \tag{5.1}$$

With our choice for the quadrature points, the previous two imply $\#\mathcal{Q}_{l,i_l} \leq 2p+1$.

We first collect all the initializations needed in Algorithm 1. It is not necessary to precompute these quantities – and in most architectures access to stored data is costly – but this used here for FLOPs evaluation.

Input: Quadrature points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}}$ as in Remark 4.1
1 **for** $l = 1, \ldots, d$ **do**
2 **for** $i_l = 1, \ldots, n_{\mathsf{DOF},l}$ **do**
3 Evaluate $\mathbb{B}^{(0)}_{l,i_l,q} := B_{i_l}(\tilde{x}_{l,q})\ \forall q \in \mathcal{Q}_{l,i_l}$, store $\mathbb{B}^{(0)}_{l,i_l} \in \mathbb{R}^{n_{\mathsf{QP},l}}$;
4 Evaluate $\mathbb{B}^{(1)}_{l,i_l,q} := B'_{i_l}(\tilde{x}_{l,q})\ \forall q \in \mathcal{Q}_{l,i_l}$, store $\mathbb{B}^{(1)}_{l,i_l} \in \mathbb{R}^{n_{\mathsf{QP},l}}$;
5 **end**
6 **for** $i_l = 1, \ldots, n_{\mathsf{DOF},l}$ **do**
7 **for** $j_l \in \mathcal{I}_{l,i_l}$ **do**
8 Calculate $\mathbb{I}^{(0,0)}_{l,i_l,j_l}$, $\mathbb{I}^{(1,0)}_{l,i_l,j_l}$, $\mathbb{I}^{(0,1)}_{l,i_l,j_l}$, $\mathbb{I}^{(1,1)}_{l,i_l,j_l}$ as defined in (4.2) ;
9 **end**
10 **end**
11 **for** $m = 1, \ldots, d$ **do**
12 Evaluate $c_{l,m}(\zeta)$ of equation (3.8) on points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}}$;
13 **end**
14 **end**
15 Evaluate $c(\zeta)$ on points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}}$;

Algorithm 1: Initializations

Then we can count operations in Algorithm 1:

(i) Evaluations of B-splines reported on lines 3–4 can be done in $\frac{1}{2}p^2$ FLOPs each. They are repeated $d\, n_{\mathsf{DOF}}\, \#\mathcal{Q}_{l,i_l}$ times so that this part costs $O\left(p^3\, n_{\mathsf{DOF}}\right)$ FLOPs.

(ii) The calculation of integrals (4.2) on line 8 needs to be done in an exact manner. The calculation of the exact integral of products of B-splines has a vast literature [34], however, closed forms are available only in some particular cases. For this reason we consider here the usual element-wise Gaussian quadrature. The evaluation of B-splines and their derivatives cost, as reported before, $\frac{1}{2}p^2$ FLOPs for each point. Counting all Gaussian point, the cost is $\approx d\ p^2$ evaluations of each of the $d\, n_{\mathsf{DOF}}$ univariate basis functions, thus costs $O(p^4\, n_{\mathsf{DOF}})$ FLOPs.

The computation of each of the integrals has the cost of a summation on $\approx p^2$ terms; and the four calculations are done $d\, n_{\mathsf{DOF}}\, \#\mathcal{I}_{l,i_l}$ times so that this costs $O\left(p^3\, n_{\mathsf{DOF}}\right)$ FLOPs.

(iii) The evaluations of the (d^2+1) functions $c_{l,m}$ and c on lines 12 and 15 have to be performed at the $N_{\mathsf{QP}} = (n_{\mathsf{QP}})^d$ quadrature points. The actual cost depends on the evaluation cost of $c_{l,m}$ and c. If these coefficients are obtained by $O(p^d)$ linear combinations of B-spline values (or derivatives), and each multivariate B-spline value is computed from multiplications of univariate B-spline values, the total cost is $C(d)\, p^d$ per quadrature point and in total $O\left(p^d\, N_{\mathsf{QP}}\right)$ FLOPs.

The leading cost of Algorithm 1 for $d \geq 2$ is $O\left(p^d\, N_{\mathsf{QP}}\right)$.

In Algorithm 2 we summarize the operations needed for the construction of the univariate WQ rules. Each calculation in lines 3–6 consists in the resolution of a linear system of dimension $\approx (2p)^2$ that is possibly underdetermined. The cost of these computations in any case negligible since it is proportional to n_{DOF}.

Input: Quadrature points $\tilde{\boldsymbol{x}}_{\boldsymbol{q}}$, B-spline evaluations $\mathbb{B}^{(\cdot)}_{l,i_l,\boldsymbol{q}}$, Integrals $\mathbb{I}^{(\cdot,\cdot)}_{l,i_l,j_l}$

for $l = 1, \dots, d$ **do**
  for $i_l = 1, \dots, n_{l,\mathsf{DOF}}$ **do**
    Calculate $w^{(0,0)}_{l,i_l,\mathcal{Q}_{l,i_l}}$ as (minimum Euclidean norm) solution of $\mathbb{B}^{(0)}_{l,\mathcal{I}_{l,i_l},\mathcal{Q}_{l,i_l}} w^{(0,0)}_{l,i_l,\mathcal{Q}_{l,i_l}} = \mathbb{I}^{(0,0)}_{l,i_l,\mathcal{I}_{l,i_l}}$;
    Calculate $w^{(1,0)}_{l,i_l,\mathcal{Q}_{l,i_l}}$ as (minimum Euclidean norm) solution of $\mathbb{B}^{(1)}_{l,\mathcal{I}_{l,i_l},\mathcal{Q}_{l,i_l}} w^{(1,0)}_{l,i_l,\mathcal{Q}_{l,i_l}} = \mathbb{I}^{(1,0)}_{l,i_l,\mathcal{I}_{l,i_l}}$;
    Calculate $w^{(0,1)}_{l,i_l,\mathcal{Q}_{l,i_l}}$ as (minimum Euclidean norm) solution of $\mathbb{B}^{(0)}_{l,\mathcal{I}_{l,i_l},\mathcal{Q}_{l,i_l}} w^{(0,1)}_{l,i_l,\mathcal{Q}_{l,i_l}} = \mathbb{I}^{(0,1)}_{l,i_l,\mathcal{I}_{l,i_l}}$;
    Calculate $w^{(1,1)}_{l,i_l,\mathcal{Q}_{l,i_l}}$ as (minimum Euclidean norm) solution of $\mathbb{B}^{(1)}_{l,\mathcal{I}_{l,i_l},\mathcal{Q}_{l,i_l}} w^{(1,1)}_{l,i_l,\mathcal{Q}_{l,i_l}} = \mathbb{I}^{(1,1)}_{l,i_l,\mathcal{I}_{l,i_l}}$;
  end
end

Algorithm 2: Construction of univariate WQ rules.

6. Formation of the mass matrix

When all the quadrature rules are available we can write the computation of the approximate mass matrix following (4.1). Similar formulae and algorithms can be written for the stiffness matrix starting from Eq. (3.2).

The mass matrix formation algorithm is mainly a loop over all rows $\boldsymbol{i}$, for each $\boldsymbol{i}$ we consider the calculation of

$$\tilde{m}_{\boldsymbol{i},\boldsymbol{j}} = \mathbb{Q}^{(0,0)}_{\boldsymbol{i}} \left(\hat{B}_{\boldsymbol{j}}(\boldsymbol{\zeta}) c(\boldsymbol{\zeta}) \right), \quad \forall \boldsymbol{j} \in \mathcal{I}_{\boldsymbol{i}}. \tag{6.1}$$

The computational cost of (6.1) is minimized by a sum factorization approach, which is explained below.

If we substitute (4.3) into (4.1) we obtain the following sequence of nested summations:

$$\tilde{m}_{\boldsymbol{i},\boldsymbol{j}} = \sum_{q_1 \in \mathcal{Q}_{1,i_1}} w^{(0,0)}_{1,i_1,q_1} \hat{B}_{j_1}(\tilde{x}_{1,q_1}) \left(\sum_{q_2 \in \mathcal{Q}_{2,i_2}} \dots \sum_{q_d \in \mathcal{Q}_{d,i_d}} w^{(0,0)}_{d,i_d,q_d} \hat{B}_{j_d}(\tilde{x}_{d,q_d}) c\left(x_{1,q_1}, \dots, x_{d,q_d}\right) \right). \tag{6.2}$$

To write (6.2) in a more compact form, we introduce the notion of matrix–tensor product [35]. Let $\mathcal{X} = \left\{x_{k_1,\dots,k_d}\right\} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ be a d-dimensional tensor, and let $m \in \{1, \dots, d\}$. The m-mode product of $\mathcal{X}$ with a matrix $A = \left\{a_{i,j}\right\} \in \mathbb{R}^{t \times n_m}$, denoted with $\mathcal{X} \times_m A$, is a tensor of dimension $n_1 \times \dots \times n_{m-1} \times t \times n_{m+1} \times \dots \times n_d$, with components

$$(\mathcal{X} \times_m A)_{k_1,\dots,k_d} = \sum_{j=1}^{n_m} a_{k_m,j}\ x_{k_1,\dots,k_{m-1},j,k_{m+1},\dots,k_d}.$$

We emphasize that such computation requires $2t \prod_{l=1}^{d} n_l$ FLOPs.

For $l = 1, \dots, d$ and $i_l = 1, \dots, n_{\mathsf{DOF},l}$ we define the matrices

$$\mathrm{B}^{(l,i_l)} = \left(\hat{B}_{j_l}(x_{l,q_l}) \right)_{j_l \in \mathcal{I}_{l,i_l}, q_l \in \mathcal{Q}_{l,i_l}}, \qquad \mathrm{W}^{(l,i_l)} = \mathrm{diag}\left(\left(w^{(0,0)}_{l,i_l,q_l} \right)_{q_l \in \mathcal{Q}_{l,i_l}} \right),$$

where $\mathrm{diag}(v)$ denotes the diagonal matrix obtained by the vector v. We also define, for each index $\boldsymbol{i}$, the d-dimensional tensor

$$\mathcal{C}_{\boldsymbol{i}} = c(\tilde{\boldsymbol{x}}_{\mathcal{Q}_{\boldsymbol{i}}}) = \left(c(\tilde{x}_{1,q_1}, \dots, \tilde{x}_{d,q_d}) \right)_{q_1 \in \mathcal{Q}_{1,i_1}, \dots, q_d \in \mathcal{Q}_{d,i_d}}.$$

Using the above notations, we have

$$\tilde{m}_{\boldsymbol{i},\mathcal{I}_{\boldsymbol{i}}} = \mathcal{C}_{\boldsymbol{i}} \times_d \left(\mathrm{B}^{(d,i_d)} \mathrm{W}^{(d,i_d)} \right) \times_{d-1} \dots \times_1 \left(\mathrm{B}^{(1,i_1)} \mathrm{W}^{(1,i_1)} \right). \tag{6.3}$$

Since with our choice of the quadrature points $\#\mathcal{Q}_{l,i_l}$ and $\#\mathcal{I}_{l,i_l}$ are both $O(p)$, the computational cost associated with (6.3) is $O(p^{d+1})$ FLOPs. Note that $\tilde{m}_{\boldsymbol{i},\mathcal{I}_{\boldsymbol{i}}}$ includes all the nonzeros entries of the $\boldsymbol{i}$th row of $\tilde{\mathbb{M}}$. Hence if we

compute it for each $\boldsymbol{i} = 1, \dots, N_{\mathsf{DOF}}$ the total cost amounts to $O(N_{\mathsf{DOF}}\, p^{d+1})$ FLOPs. This approach is summarized in Algorithm 3.

We remark that writing the sums in (6.2) in terms of matrix–tensor products as in (6.3) is very useful from an implementation viewpoint. Indeed, in interpreted languages like MATLAB (which is the one used in the experiments of the next section), it is crucial to avoid loops and vectorize (in our case, tensorize) the operations, in order to obtain an efficient implementation of an algorithm; see also the discussion in [36]. In particular, each matrix–tensor product in (6.3) is computed via a simple matrix–matrix product, which is a BLAS level 3 operation and typically yields high efficiency on modern computers.

Input: Quadrature rules, evaluations of coefficients
for $\boldsymbol{i} = 1, \dots, N_{\mathsf{DOF}}$ **do**
  Set $\mathcal{C}_{\boldsymbol{i}}^{(0)} := c(\tilde{\boldsymbol{x}}_{\mathcal{Q}_{\boldsymbol{i}}})$;
  for $l = d, d-1, \dots, 1$ **do**
    Load the quadrature rule $\mathbb{Q}_{i_l}^{(0,0)}$ and form the matrices $\mathrm{B}^{(l,i_l)}$ and $\mathrm{W}^{(l,i_l)}$;
    Compute $\mathcal{C}_{\boldsymbol{i}}^{(d+1-l)} = \mathcal{C}_{\boldsymbol{i}}^{(d-l)} \times_l \left(\mathrm{B}^{(l,i_l)}\mathrm{W}^{(l,i_l)}\right)$;
  end
  Store $\tilde{m}_{\boldsymbol{i},\mathcal{I}_{\boldsymbol{i}}} = \mathcal{C}_{\boldsymbol{i}}^{(d)}$;
end

Algorithm 3: Construction of mass matrix by sum-factorization.

7. Numerical tests

In this section, in order to evaluate numerically the behaviour of the proposed procedure we present some numerical tests. First, in Section 7.1 we consider the solution of a 1D problem where we see that the application of our row-loop WQ-based algorithm leads to optimal order of convergence. Then, in Section 7.2 we measure the performance of the algorithm. We consider there the formation of mass matrices in 3D. The results for all cases refer to a Linux workstation equipped with Intel i7-5820K processors running at 3.30 GHz, and with 64 GB of RAM. The row-loop WQ-based algorithm is potentially better suited for a parallel implementation than the standard element-wise SGQ-based algorithm, however we benchmark here sequential execution and use only one core for the simulations.

7.1. Convergence of approximate solution in 1D

As a test with known solution we consider the following:

$$\begin{cases} u'' + u(x) = \dfrac{5\exp(2x) - 1}{4} & \text{on} \quad [0, \pi/6] \\ u(0) = 0,\, u(\pi/6) = \exp(\pi/3)/2. \end{cases} \tag{7.1}$$

We compare the numerical solution in the parametric domain, using the geometric transformation $t = 2\sin(x)$, with the exact one $u(x) = \frac{\exp(2x)-1}{4}$. Then we calculate point-wise absolute error, integral error and energy error – namely L^∞, L^2 and H^1 norms – with varying spline degree p. Fig. 4 illustrates that the construction of the matrices with the proposed procedure does not effect the overall convergence properties of the Galerkin method, as it can be seen by comparing the convergence curves with those obtained using Gaussian quadrature.

Remark 7.1 (*Convergence Rates*)**.** As already noted in Section 2, WQ does not preserve symmetry, that is in general $\tilde{m}_{i,j} \neq \tilde{m}_{j,i}$ even if $m_{i,j} = m_{j,i}$. The lack of symmetry did not cause any deterioration of the order of convergence in energy and lower-order norms in our numerical benchmarking, see Fig. 4. This is an important and interesting behaviour that deserves further study. We remark that the lack of symmetry occurs also for collocation isogeometric schemes [37], where however convergence rates are suboptimal.

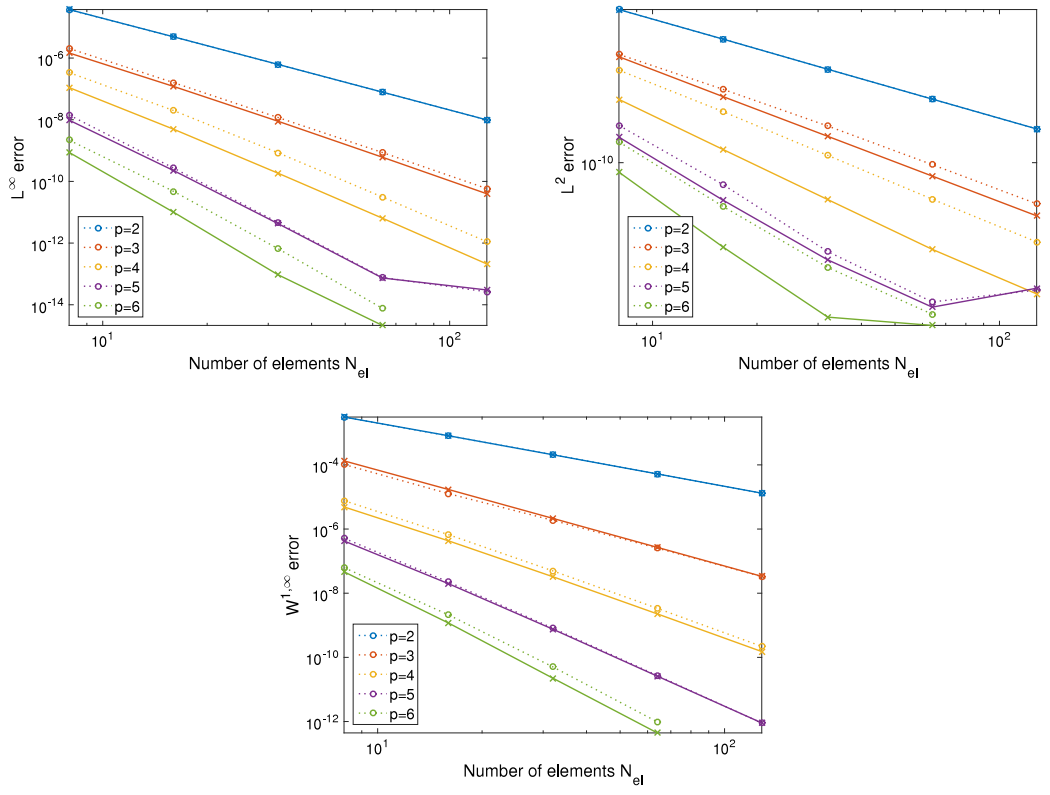

Fig. 4. Convergence history plot. We report errors in L^∞, L^2 and H^1 norm for the solution of problem (7.1) by Galerkin based isogeometric analysis with WQ (Algorithms 1–3) for various degrees p in dotted lines. As reference, the solid lines refer to the same calculation made with element-wise SGQ. Optimal convergence rate is achieved in all cases. SGQ is slightly more accurate for even degrees > 2 in L^2 and L^∞ norms.

7.2. *Time for the formation of matrices*

In this section we report CPU time results for the formation on a single patch domain of mass matrices. Comparison is made with GeoPDEs, the optimized but SGQ-based MATLAB isogeometric library developed by Rafael Vázquez, see [26,27]. In Fig. 5 we plot the time needed for the mass matrix formation up to degree $p = 10$ with $N_{\mathsf{DOF}} = 20^3$. The tests confirm the superior performance of the proposed row-loop WQ-based algorithm vs SGQ. In the case $p = 10$ GeoPDEs takes more than 62 h to form the mass matrix while the proposed algorithm needs only 27 s, so that the use of high degrees is possible with WQ.

Remark 7.2 (*Sparse Implementation of WQ*)**.** Clearly we exploit sparsity in our MATLAB implementation: we compute all the nonzero entries of $\tilde{\mathbb{M}}$, the corresponding row and column indices and then call the MATLAB `sparse` function, that uses a compressed sparse column format.

In the last test, we experimentally study the growth order of the computational effort needed to form $\tilde{\mathbb{M}}$, and we highlight which parts of the code mainly contributes to this effort.

In Fig. 6, we plot in a log–log scale the total computation time spent by Algorithms 1–3 for 40^3 elements and spline degree up to 10. We also plot the time spent in the computations of the matrix–tensor products (i.e., line 5 of Algorithm 3, which is the dominant step with respect to the number of FLOPs of the whole procedure), and the time used by the MATLAB function `sparse`, which is responsible of allocating the memory for $\tilde{\mathbb{M}}$ and copying the entries in the sparse matrix data structure. These timings were obtained using the profiler of MATLAB.[2] If we consider the products time, we can see that the growth relative to p is significantly milder than what is indicated by the theoretical FLOP

[2] The same timings were also computed using the commands `tic` and `toc`, yielding similar results.

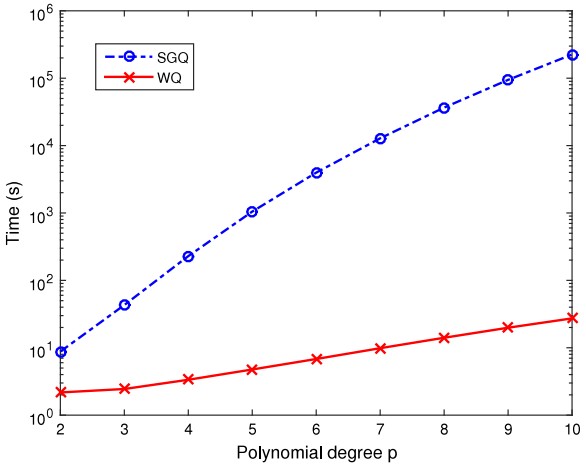

Fig. 5. Time for mass matrix assembly in the framework of isogeometric-Galerkin method with maximal regularity on a single patch domain of 20^3 elements. The comparison is between the WQ approach proposed (Algorithms 1–3) and SGQ as implemented in GeoPDEs 3.0.

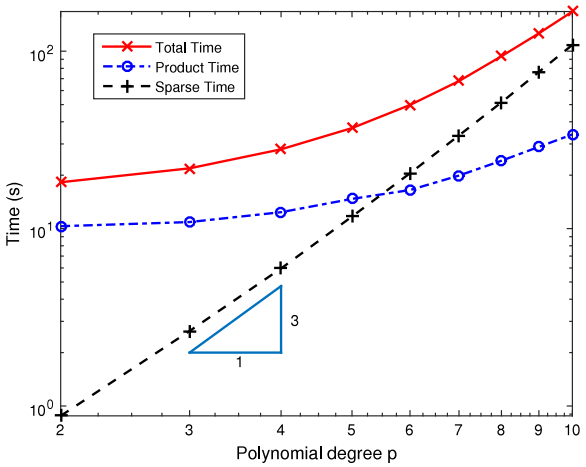

Fig. 6. Time for mass matrix formation with the WQ approach proposed in this paper. In this case $d = 3$, $n_{el} = 40^3$. Reference slope is p^3. Along with the total time, we show the time spent by the product (6.3) and by the function `sparse`, which represent the single most relevant computational efforts of our code. Other timings, which become negligible for large p, are not shown.

counting, i.e., $O(N_{\mathsf{DOF}}\, p^4)$. This is probably related to the small dimension of the matrices and tensors involved. On the other hand, the times spent by the `sparse` function is clearly proportional to p^3, as highlighted in the plot by a reference triangle with slope 3. This is expected, as the number of nonzero entries of $\tilde{\mathbb{M}}$ is $O(N_{\mathsf{DOF}}\, p^3)$. What is surprising is that, for $p > 5$ the time of the `sparse` call dominates the total time of the algorithm. This indicates that our approach is in practice giving the best possible performance at least for degree high enough, since the `sparse` call is unavoidable and well optimized in MATLAB.

Furthermore, the computing time depends linearly on N_{DOF}, as expected, but for brevity we do not show the results.

8. Conclusions

The proposed algorithm for the formation of isogeometric Galerkin matrices is based on three concepts. First, we use a row loop instead of an element loop. Second, we use WQ that gives significant savings in quadrature points. Third, we exploit the tensor-product structure of the B-spline basis functions, adopting an optimized sum-factorization implementation as in [6]. Our approach also incorporates an idea of a previous work: the numerical computation of univariate quadrature rules as in [15] and following papers. In the present work, however, we fix a priori the quadrature

points so that the weights are given by solving a linear problem, and we use sum-factorization cycling on rows and not on elements. The result is a significant gain in performance compared to standard approaches, for all polynomial degrees but especially for high degree. For example, in the numerical tests that we present, for $p = 6$ the time of formation of a mass matrix is seconds vs hours (comparison made with GeoPDEs 3.0, which has a well optimized but standard design, see [26,27]), where in our algorithm the computational time is dominated by the unavoidable MATLAB `sparse` function call. These results pave the way to the practical use of high-degree k-refinement. Moreover they relight the interest for a comparison between Galerkin and collocation formulation, that is nowadays preferred for high-degree isogeometric simulations, see [37]. Curiously, a Galerkin formulation with WQ is closer to collocation, from the viewpoint of the computational cost and since both do not preserve symmetry, i.e. the matrices formed from symmetric differential operators are not symmetric. However WQ should preserve the other main properties of Galerkin formulations.

Our work will continue in three different directions. We need to develop a full mathematical analysis of this approach. We will work on a full implementation within GeoPDEs. Finally, we will develop the proposed approach in the direction of non-tensor product spaces (T-splines, hierarchical splines, etc.), where we expect that some significant advantages of our approach will be maintained.

Acknowledgements

The authors would like to thank Rafael Vázquez for fruitful discussions on the topic of the paper. Francesco Calabrò was partially supported by INdAM, through GNCS research projects. Giancarlo Sangalli and Mattia Tani were partially supported by the European Research Council through the FP7 ERC Consolidator Grant No.616563 *HIGEOM*, and by the Italian MIUR through the PRIN "Metodologie innovative nella modellistica differenziale numerica". This support is gratefully acknowledged.

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Engrg. 194 (39) (2005) 4135–4195.
[2] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, John Wiley & Sons, 2009.
[3] Y. Bazilevs, L. Beiraõ da Veiga, J.A. Cottrell, T.J.R. Hughes, G. Sangalli, Isogeometric analysis: approximation, stability and error estimates for h-refined meshes, Math. Models Methods Appl. Sci. 16 (07) (2006) 1031–1090.
[4] L. Beirão da Veiga, A. Buffa, J. Rivas, G. Sangalli, Some estimates for h–p–k-refinement in isogeometric analysis, Numer. Math. 118 (2) (2011) 271–305.
[5] L. Beirão da Veiga, A. Buffa, G. Sangalli, R. Vázquez, Mathematical analysis of variational isogeometric methods, Acta Numer. 23 (2014) 157–287.
[6] P. Antolin, A. Buffa, F. Calabrò, 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.
[7] G. Sangalli, M. Tani, Isogeometric preconditioners based on fast solvers for the Sylvester equation, 2016, pp. 1–28. arXiv preprint arXiv:1602.01636.
[8] A. Mantzaflaris, B. Jüttler, Exploring matrix generation strategies in isogeometric analysis, in: Mathematical Methods for Curves and Surfaces, Springer, 2014, pp. 364–382.
[9] D. Rypl, B. Patzák, Study of computational efficiency of numerical quadrature schemes in the isogeometric analysis, Eng. Mech. (2012) 304.
[10] M. Ainsworth, G. Andriamaro, O. Davydov, Bernstein-Bézier finite elements of arbitrary order and optimal assembly procedures, SIAM J. Sci. Comput. 33 (6) (2011) 3087–3109.
[11] M. Ainsworth, O. Davydov, L.L. Schumaker, Bernstein-Bézier finite elements on tetrahedral hexahedral pyramidal partitions, Comput. Methods Appl. Mech. Engrg. 304 (2016) 140–170.
[12] C. Adam, T.J.R. 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.
[13] D. Schillinger, S.J. Hossain, T.J.R. Hughes, Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 277 (2014) 1–45.
[14] M. Hillman, J.S. Chen, Y. Bazilevs, Variationally consistent domain integration for isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 284 (2015) 521–540.
[15] T.J.R. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 199 (5) (2010) 301–313.
[16] J. Bremer, Z. Gimbutas, V. Rokhlin, A nonlinear optimization procedure for generalized Gaussian quadratures, SIAM J. Sci. Comput. 32 (4) (2010) 1761–1788.
[17] H. Cheng, V. Rokhlin, N. Yarvin, Nonlinear optimization, quadrature, and interpolation, SIAM J. Optim. 9 (4) (1999) 901–923.

[18] J. Ma, V. Rokhlin, S. Wandzura, Generalized Gaussian quadrature rules for systems of arbitrary functions, SIAM J. Numer. Anal. 33 (3) (1996) 971–996.
[19] R. Ait-Haddou, M. Bartoň, V.M. Calo, Explicit Gaussian quadrature rules for C^1 cubic splines with symmetrically stretched knot sequences, J. Comput. Appl. Math. 290 (2015) 543–552.
[20] M. Bartoň, V.M. Calo, Gaussian quadrature for splines via homotopy continuation: Rules for C^2 cubic splines, J. Comput. Appl. Math. 296 (2016) 709–723.
[21] 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.
[22] K.A. Johannessen, Optimal quadrature for univariate and tensor product splines, Comput. Methods Appl. Mech. Engrg. (2016) http://dx.doi.org/10.1016/j.cma.2016.04.030.
[23] F. Auricchio, F. Calabrò, T.J.R. 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.
[24] A. Mantzaflaris, B. Jüttler, Integration by interpolation and look-up for Galerkin-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 284 (2015) 373–400.
[25] A. Mantzaflaris, B. Jüttler, B. Khoromskij, U. Langer, Matrix generation in isogeometric analysis by low rank tensor approximation, in: Curves and Surfaces, Springer, 2015, pp. 321–340.
[26] C. De Falco, A. Reali, R. Vázquez, Geopdes: a research tool for isogeometric analysis of pdes, Adv. Eng. Softw. 42 (12) (2011) 1020–1034.
[27] R. Vázquez, A new design for the implementation of isogeometric analysis in Octave and Matlab: GeoPDEs 3.0, Tech. Report 16-02, IMATI Report Series, April 2016.
[28] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, SIAM, 2002.
[29] G. Strang, G.J. Fix, An Analysis of The Finite Element Method, vol. 212, Prentice-Hall Englewood Cliffs, NJ, 1973.
[30] A. Karatarakis, P. Karakitsios, M. Papadrakakis, Gpu accelerated computation of the isogeometric analysis stiffness matrix, Comput. Methods Appl. Mech. Engrg. 269 (2014) 334–355.
[31] F. Calabrò, C. Manni, F. Pitolli, Computation of quadrature rules for integration with respect to refinable functions on assigned nodes, Appl. Numer. Math. 90 (2015) 168–189.
[32] L. Schumaker, Spline Functions: Basic Theory, Cambridge Mathematical Library, 2007.
[33] C. De Boor, A Practical Guide to Splines, Springer, 1978.
[34] A.H. Vermeulen, R.H. Bartels, G.R. Heppler, Integrating products of B-splines, SIAM J. Sci. Stat. Comput. 13 (4) (1992) 1025–1038.
[35] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.
[36] F. Cuvelier, C. Japhet, G. Scarella, An efficient way to assemble finite element matrices in vector languages, BIT 56 (3) (2015) 1–32.
[37] D. Schillinger, J. A Evans, A. Reali, M.A. Scott, T.J.R. Hughes, Isogeometric collocation: Cost comparison with Galerkin methods and extension to adaptive hierarchical nurbs discretizations, Comput. Methods Appl. Mech. Engrg. 267 (2013) 170–232.