NFN - Nationales Forschungsnetzwerk

Geometry + Simulation

http://www.gs.jku.at

Fast Formation of Isogeometric Galerkin Matrices via Integration by Interpolation and Look-up

Maodong Pan, Bert Jüttler,
Alessandro Giust

G+S Report No. 88

March 2020

Der Wissenschaftsfonds.

JOHANNES KEPLER
UNIVERSITY LINZ

Fast Formation of Isogeometric Galerkin Matrices via Integration by Interpolation and Look-up

Maodong Pan[a], Bert Jüttler[a,b,*], Alessandro Giust[b]

[a] *Institute of Applied Geometry, Johannes Kepler University, Linz, Austria*
[b] *Doctoral Program "Computational Mathematics", Johannes Kepler University, Linz, Austria*

Abstract

Although isogeometric analysis possesses many advantages over classical finite element methods, the computational costs of matrix assembly (especially for high polynomial degrees) constitute a bottleneck in isogeometric numerical simulations. To address this issue, we propose an efficient algorithm for the formation of isogeometric Galerkin matrices based on the interpolation, look-up and sum factorization techniques. This method consists of three steps: First, we project the common factors occurring in the integrals into an appropriate spline space via the interpolation or quasi-interpolation operator. Subsequently, the entries of the stiffness and mass matrices are approximated by a sum of integrals of tensor-product B-splines. Second, to perform an exact integration of the integrands in the approximated matrices, a look-up table for the standardized B-spline tri-product integrals is built. Finally, the system matrices are efficiently assembled by invoking the sum factorization technique. We present a detailed analysis of the computational costs, in order to compare the new method with the existing approaches for all polynomial degrees. Several numerical tests confirm that the proposed method ensures the efficiency of matrix assembly. In addition, the extension of our approach to matrix-free applications is also discussed.

Keywords: Isogeometric analysis; Matrix assembly; Computational complexity; Interpolation and look-up; Sum factorization

1. Introduction

Isogeometric analysis (IgA), introduced by Hughes et al. [26], provides a true design-through-analysis framework by employing the same mathematical representations (such as NURBS) for both the geometry description and the discretization of partial differential equations (PDEs) [16]. Consequently, it possesses significant advantages over the traditional finite element method. These include higher regularity of the isogeometric discretizations of PDEs [10], enhanced robustness in a number of practical applications [25], and improved accuracy of the obtained numerical solutions [17, 18].

*Corresponding author.
Email addresses: mdpan@mail.ustc.edu.cn (Maodong Pan), bert.juettler@jku.at (Bert Jüttler)

However, it has been noted that these advantages come at the price of increased computational complexity per degree of freedom in the matrix assembly process, and this effect is more pronounced for high spatial dimensions and spline degrees [13, 38]. In particular, the process of assembling the stiffness and mass matrices via standard element-wise Gaussian quadrature leads to complexity of $\mathcal{O}(Np^{3d})$, where N, p, d denote the number of degrees of freedom, the spline degree and the spatial dimension respectively. It is obvious that the computational cost grows fast as p increases, especially for $d = 3$. Techniques for the fast formation of system matrices became an active topic in isogeometric analysis, and substantial research effort has been devoted to efficient algorithms for generating the stiffness and mass matrices arising in isogeometric discretizations.

The work on this topic can be roughly classified into four categories: (A) reduced or specialized quadrature rules; (B) isogeometric collocation methods; (C) quadrature techniques employing sum factorization; (D) quadrature-free approaches.

(A) Reduced or specialized quadrature rules. Initially, Gaussian quadrature was widely used for isogeometric simulation, but its performance is far from optimal. This is attributed to the fact that Gaussian quadrature rules do not take the smoothness of NURBS basis functions across element boundaries into account, resulting in more quadrature points than required for stability and accuracy. Thus one way to improve the computational efficiency of numerical integration is to reduce the number of quadrature nodes. This was first proposed by Hughes et al. [27] through particular instances and the development of quadrature rules for NURBS patches with uniform knots. Subsequently, a large number of related approaches [1, 4, 6–9, 14, 15, 22, 28, 42] focusing on deriving optimal (with respect to the number of evaluations) quadrature rules for various spline spaces have been explored. Among these approaches, the most advanced one is weighted quadrature [15], which is a sophisticated strategy to exploit sum factorization. The idea is to design specialized quadrature rules that require fewer evaluations, thereby generating a significant speedup with complexity $\mathcal{O}(Np^{d+1})$. Various important issues concerning the practical implementation of this method are discussed in the recent work [23]. While weighted quadrature is efficient, it is not able to preserve the symmetry of the resulting system matrices.

(B) Isogeometric collocation methods. Isogeometric collocation, originally presented in [5], is a valuable alternative approach to the Galerkin method, in which the strong form of the underlying PDEs is enforced at a set of locations (called collocation points), based on the higher regularity of the isogeometric discretizations. The main advantage of isogeometric collocation over Galerkin methods is the minimal computational effort of $\mathcal{O}(Np^d)$ for matrix assembly, since for each degree of freedom only one evaluation (at the associated collocation point) is required [41]. Thus collocation methods can be considered as one-point quadrature. A disadvantage is the lack of theoretical guarantees for the convergence properties. To address this issue, an ideal collocation scheme whose solution coincides with the solution of the Galerkin method was introduced in [21], thus recovering optimal rate of convergence. Within this framework, an improved convergence behaviour can be obtained by using a subset of the collocation points estimated in [21] with local symmetry [35]. The investigation

of super-convergent points also contributes to the development of special quadrature rules for IgA [19]. Despite the recent progress, a rigorous mathematical explanation for the convergence behaviour observed for these methods is not available yet.

(C) Quadrature techniques employing sum factorization. Sum factorization, a well-known technique in the field of spectral elements and high-order finite elements [2, 34, 36], was first employed for assembling isogeometric Galerkin matrices in [3]. In this initial contribution to the use of sum factorization in IgA, the assembly process was performed in an element-wise procedure and has a computational complexity $\mathcal{O}(Np^{2d+1})$. Four years later, Bressan and Takacs [12] applied the sum factorization technique in a global fashion, thereby reducing the complexity to $\mathcal{O}(Np^{d+2})$. In fact, all the quadrature rules with tensor-product structures admit the acceleration via the sum-factorization technique.

(D) Quadrature-free approaches. These methods abandon the use of numerical integration altogether. In the interpolation and look-up approach [32], a factor in the integrand composed of both the coefficients of the PDEs as well as the geometry mapping is first projected into a spline space, and subsequently the resulting integrals are evaluated exactly via precomputed compact look-up tables. The total complexity of this method is $\mathcal{O}(Np^{2d})$. Another category of quadrature-free work is towards exploitation of the tensor methods. It has been observed that the multivariate functional integrals occurring in isogeometric discretizations can be well approximated by a sum of small number of univariate functional integrals with the use of tensor decomposition technique. In [33], the authors have explored this technique to perform the assembly task of isogeometric matrices. To avoid the computationally expensive higher-order singular value decomposition required in the tensor decomposition step, Scholz et al. [43] employed standard singular value decomposition to decouple isogeometric discretizations partially, while maintaining a quasi-optimal complexity for the matrix formation. For these low-rank assembling approaches, the overall computational complexity is $\mathcal{O}(NRp^d)$, where R is the approximated rank of the tensors representing the quadrature weight function. A related method that exploits the tensor-product structure (more precisely, the low Kronecker rank of the matrices) was established recently [24].

Summing up, the fastest available methods for isogeometric matrix assembly are weighted quadrature (with complexity $\mathcal{O}(Np^{d+1})$) and the low-rank assembling approach (with complexity $\mathcal{O}(NRp^d)$). However, the first method does not keep the symmetry of the matrices, and the second one requires tensor decomposition methods. Moreover, the performance of the low-rank assembling approach is determined by the tensor rank R, which depends on the geometry and on the coefficients of the PDEs.

The present paper describes an efficient method for the assembling of isogeometric Galerkin matrices, which has the complexity $\mathcal{O}(Np^{d+1})$. The main contributions are:

- We combine the techniques of interpolation, look-up and sum factorization, thereby introducing a fast matrix assembly algorithm.
- A detailed analysis of the computational costs in terms of the asymptotic number of floating point operations per degree of freedom for different matrix assembly approach-

es is provided. This confirms that the new method also performs well for low degree p.

- In addition to matrix assembly, the extension of our method for matrix-free applications is also presented.

The remainder of the paper is structured as follows. In the following section, we introduce the model problem and recall some facts concerning Galerkin-based discretizations in IgA. Sections 3–7 present our approach in detail. In Section 8, several experimental results are demonstrated to verify the effectiveness of the proposed method, and comparisons with other approaches are also provided. In Section 9, the extension of our method to matrix-free applications is presented, and the subsequent section discusses the proposed method for assembling the stiffness matrix. The paper concludes with a summary and future work. Additional details concerning interactions between B-splines, the complexities of other techniques for matrix assembly and other methods for matrix-free applications are postponed to three appendices.

2. Preliminaries

As our model problem, we consider the reaction-diffusion equation with homogeneous Dirichlet boundary conditions

$$\begin{cases} -\mathrm{div}(D\nabla u) + \lambda u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{1}$$

on the domain $\Omega \subset \mathbb{R}^d$ ($d \leq 3$), where $D = D(\boldsymbol{x})$ is a symmetric and uniformly positive definite matrix, and $\lambda = \lambda(\boldsymbol{x})$ is a non-negative function. The variational form of this problem consists in finding $u \in H_0^1(\Omega)$, such that

$$a(u, v) = f(v), \quad \forall v \in H_0^1(\Omega), \tag{2}$$

where

$$a(u, v) = \int_\Omega \nabla u^T(\boldsymbol{x}) D(\boldsymbol{x}) \nabla v(\boldsymbol{x}) + \lambda(\boldsymbol{x}) u(\boldsymbol{x}) v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}, \quad f(v) = \int_\Omega f(\boldsymbol{x}) v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}.$$

In isogeometric analysis, the computational domain Ω is parameterized by a geometry mapping

$$\boldsymbol{F} : \hat{\Omega} \to \Omega$$

which is typically represented by NURBS. In order to keep the presentation simple, we will assume that this mapping is a d-variate single-patch tensor-product spline function

$$\boldsymbol{F}(\hat{\boldsymbol{x}}) = \sum_{\boldsymbol{i} \in \mathcal{I}} \boldsymbol{c_i} \hat{\beta}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}),$$

which is defined on the parameter domain $\hat{\Omega} = [0,1]^d$. The mapping is specified by control points $\boldsymbol{c_i}$, which are multiplied by tensor-product B-splines

$$\hat{\beta}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}) = \prod_{\ell=1}^{d} \hat{\beta}_{\ell,i_\ell}(\hat{x}_\ell), \quad \boldsymbol{i} = (i_1, \ldots, i_d)$$

of degree p. These multivariate B-splines $\hat{\beta}_{\boldsymbol{i}}$ are products of the d univariate B-splines $\hat{\beta}_{\ell,i_\ell}$ of degree p defined on $[0,1]$.

For simplicity, it will be assumed that both the geometry mapping and the isogeometric discretization use n univariate B-splines of degree p and smoothness C^{p-1} in each coordinate direction. Consequently, all inner knots possess multiplicity 1, and these knots subdivide each of the univariate domains into $n-p$ univariate elements. The finite index set $\mathcal{I}$ thus takes the form

$$\mathcal{I} = \{\boldsymbol{i} \ : \ 0 < i_\ell \leq n, \forall\ \ell = 1, \ldots, d\}.$$

The isogeometric discretization takes advantages of the given parameterization of the domain Ω. Specifically, the discretization space $V_h \subset H_0^1(\Omega)$ is defined by

$$V_h = \operatorname{span}\{\phi_{\boldsymbol{i}} : \phi_{\boldsymbol{i}} = \hat{\beta}_{\boldsymbol{i}} \circ \boldsymbol{F}^{-1},\ \phi_{\boldsymbol{i}}|_{\partial\Omega} = 0,\ \boldsymbol{i} \in \mathcal{I}\}.$$

The finite-dimensional space V_h is used for isogeometric approximation of the weak form in (2), which consists in finding $u_h \in V_h$, such that

$$a(u_h, v_h) = f(v_h), \quad \forall v_h \in V_h. \tag{3}$$

The approximate solution u_h is written as

$$u_h = \sum_{\boldsymbol{i} \in \mathcal{I}} u_{\boldsymbol{i}} \phi_{\boldsymbol{i}} \tag{4}$$

with unknown coefficients $\boldsymbol{u} = \{u_{\boldsymbol{i}}\}_{\boldsymbol{i} \in \mathcal{I}}$. We substitute this expansion into (3) and take $v_h = \phi_{\boldsymbol{i}},\ \boldsymbol{i} \in \mathcal{I}$. The coefficients of the approximate solution are found by solving the linear system

$$(S + M)\boldsymbol{u} = \boldsymbol{b}, \tag{5}$$

which involves the stiffness matrix S, the mass matrix M and the load vector $\boldsymbol{b}$. The elements of these quantities take the form

$$S_{\boldsymbol{ij}} = \int_\Omega \nabla \phi_{\boldsymbol{i}}^T D \nabla \phi_{\boldsymbol{j}} \mathrm{d}\boldsymbol{x} = \int_{\hat{\Omega}} \hat{\nabla} \hat{\beta}_{\boldsymbol{i}}^T \widetilde{D} \hat{\nabla} \hat{\beta}_{\boldsymbol{j}} \mathrm{d}\hat{\boldsymbol{x}}, \tag{6}$$

$$M_{\boldsymbol{ij}} = \int_\Omega \lambda \phi_{\boldsymbol{i}} \phi_{\boldsymbol{j}} \mathrm{d}\boldsymbol{x} = \int_{\hat{\Omega}} \hat{\lambda} \hat{\beta}_{\boldsymbol{i}} \hat{\beta}_{\boldsymbol{j}} |\det \hat{\nabla} \boldsymbol{F}| \mathrm{d}\hat{\boldsymbol{x}}, \tag{7}$$

and

$$b_{\boldsymbol{i}} = \int_\Omega f \phi_{\boldsymbol{i}} \mathrm{d}\boldsymbol{x} = \int_{\hat{\Omega}} \hat{f} \hat{\beta}_{\boldsymbol{i}} |\det \hat{\nabla} \boldsymbol{F}| \mathrm{d}\hat{\boldsymbol{x}} \tag{8}$$

respectively, where

$$\widetilde{D} = \hat{\nabla}\boldsymbol{F}^{-1}\hat{D}\hat{\nabla}\boldsymbol{F}^{-T}|\det\hat{\nabla}\boldsymbol{F}|$$

is a $d \times d$ matrix, and $\hat{D} = D \circ \boldsymbol{F}$, $\hat{\lambda} = \lambda \circ \boldsymbol{F}$, $\hat{f} = f \circ \boldsymbol{F}$. Moreover, the symbol $\hat{\nabla}$ denotes the gradient operator in the parameter domain $\hat{\Omega}$.

Remark 1. Similar to the previous works [3, 15], we assume the physical domain is parameterized by a single-patch tensor-product spline function just for the sake of simplicity, althogh it is well known that NURBS are typically employed for representing both the computational domain and numerical solution in isogeometric analysis. In fact, the proposed approach can be extended to the case of using NURBS as the basis (see [32, Remark 4] and [43, Section 2.3]), since this situation only leads to slightly involved formulas for the quantities $\widetilde{D}$, $\hat{\lambda}|\det\hat{\nabla}\boldsymbol{F}|$ and $\hat{f}|\det\hat{\nabla}\boldsymbol{F}|$ appearing in the stiffness matrix (6), mass matrix (7) and load vector (8) respectively. In addition, non-rational splines are sufficient to represent the geometry in many cases. ◇

3. Outline of the method

In order to keep the presentation concise, we will focus on the mass matrix

$$M_{\boldsymbol{ij}} = \int_{\hat{\Omega}} \hat{\beta}_{\boldsymbol{i}}\hat{\beta}_{\boldsymbol{j}} w \mathrm{d}\hat{\boldsymbol{x}}, \quad w = \hat{\lambda}|\det\hat{\nabla}\boldsymbol{F}|. \tag{9}$$

The stiffness matrix (6) and load vector (8) can be dealt with similarly. We describe our method and analyze the computational complexity in the three-dimensional case ($d = 3$).

The difficulties for the evaluation of the integrals (9) are caused by the function w, which is generally non-polynomial and possibly of high degree. Moreover, this function is a common factor shared by many integrals. Unlike the quadrature-based methods, we propose to replace w by a spline (quasi-) interpolation, followed by an exact and efficient evaluation of the resulting integrals. The overall pipeline of our approach can be outlined as follows:

- **Step I:** First we apply the interpolation or quasi-interpolation operator to project the common factor w into a tensor-product spline space.
- **Step II:** Following the spline projection, the entries of the mass matrix can be approximated by a sum of elementary integrals of tri-product B-splines. In order to allow for an exact integration of the integrands in the approximated mass matrix, a look-up table for the standardized B-spline tri-product integrals is built.
- **Step III:** Finally the matrix is efficiently assembled by invoking the sum factorization technique.

In the following sections, we will discuss the three steps in detail.

4. Step I: Spline projection by interpolation or quasi-interpolation

First we project the function w into the tensor-product spline space $\mathbb{S} = \text{span}\{\hat{\beta}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{I}\}$ containing spline functions of degree p that are C^{p-1} smooth, which is also used to define the isogeometric discretization. As observed in [32], choosing this space allows to maintain the optimal order of approximation. Applying a spline projection (e.g., an interpolation or quasi-interpolation operator) to w, we arrive at

$$w(\hat{\boldsymbol{x}}) \approx \sum_{\boldsymbol{k} \in \mathcal{I}} w_{\boldsymbol{k}} \hat{\beta}_{\boldsymbol{k}}(\hat{\boldsymbol{x}}) \tag{10}$$

with the coefficients $w_{\boldsymbol{k}}$.

It should be noted that the spline projection via interpolation or quasi-interpolation is the only step that introduces a numerical error in the evaluation of the integrals (9). The influence of this error to the overall accuracy has been analyzed in [32] based on Strang's lemma. It was shown that a spline approximation of degree p suffices to preserve the overall accuracy of the isogeometric simulation.

For the *interpolation* operator, the approximation of w is constructed by tensor-product spline interpolation at Greville abscissas, and the coefficients of the interpolant can be obtained using de Boor's algorithm [11]. A detailed description of this procedure was given in [32].

The use of *quasi-interpolation* provides a valuable alternative approach. The quasi-interpolant of a univariate function $\varphi(\hat{x})$ by B-spline basis $\{\hat{\beta}_i : i = 1, \ldots, n\}$ of degree p takes the general form

$$Q^p \varphi(\hat{x}) = \sum_{i=1}^{n} \mu_i^p(\varphi)\, \hat{\beta}_i(\hat{x}).$$

Various choices of the linear coefficient functionals μ_i^p have been proposed in the literature, see e.g. [39] for the uniform case. We focus on coefficient functionals that are based on values of φ at certain sample points, taking the form

$$\mu_i^p(\varphi) = \sum_j \alpha_{i,j} \varphi(\gamma_j). \tag{11}$$

Appendix B recalls a simple approach invented by Lyche and Schumaker [31] for quasi-interpolation with non-uniform knots. The two plots in Figure 1 compare interpolation and quasi-interpolation by quadratic splines. Both methods are comparable, but interpolation has a slightly better accuracy.

The extension of quasi-interpolation to tensor-product splines is straightforward: E.g., for dimension $d = 3$, each of the n^3 coefficients in (10) is computed from at most $(p+2)^3$ sampled values $w(\gamma_{1,j_1}, \gamma_{1,j_2}, \gamma_{1,j_3})$, which can be expressed as

$$\begin{aligned} w_{\boldsymbol{k}} &= \sum_{\boldsymbol{j}} \alpha_{\boldsymbol{kj}} w(\gamma_{1,j_1}, \gamma_{2,j_2}, \gamma_{3,j_3}) \\ &= \sum_{j_3} \alpha_{3,k_3 j_3} \sum_{j_2} \alpha_{2,k_2 j_2} \sum_{j_1} \alpha_{1,k_1 j_1} w(\gamma_{1,j_1}, \gamma_{2,j_2}, \gamma_{3,j_3}), \end{aligned} \tag{12}$$

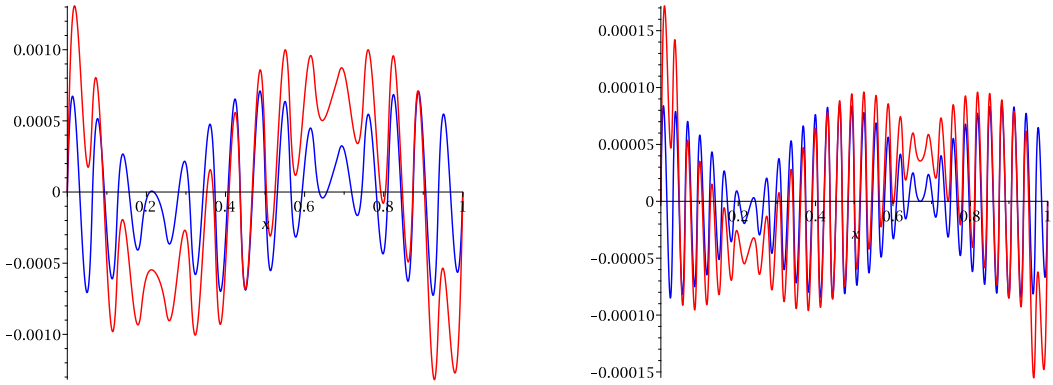


Figure 1: Error of two spline projections of $\varphi(\hat{x}) = \sin 7\hat{x}$ with 16 (left) and 32 (right) knot spans. The splines are obtained via interpolation (blue) and quasi-interpolation (red).

where the weights $\alpha_{\boldsymbol{kj}} = \alpha_{1,k_1 j_1}\alpha_{2,k_2 j_2}\alpha_{3,k_3 j_3}$ are determined by the coefficient functionals μ_i^p, possessing a tensor-product structure that facilitates the use of the sum factorization approach.

5. Step II: Approximation of the integral by spline projection, look-up tables

Using the spline approximation of w in (10), we approximate the matrix elements by sums of elementary integrals of tensor-product B-splines

$$M_{\boldsymbol{ij}} \approx \int_{\hat{\Omega}} \hat{\beta}_{\boldsymbol{i}}\hat{\beta}_{\boldsymbol{j}} \sum_{\boldsymbol{k}\in\mathcal{I}} w_{\boldsymbol{k}}\hat{\beta}_{\boldsymbol{k}} \mathrm{d}\hat{\boldsymbol{x}} = \sum_{\boldsymbol{k}\in\mathcal{I}} w_{\boldsymbol{k}} \int_{\hat{\Omega}} \hat{\beta}_{\boldsymbol{i}}\hat{\beta}_{\boldsymbol{j}}\hat{\beta}_{\boldsymbol{k}} \mathrm{d}\hat{\boldsymbol{x}}. \tag{13}$$

By introducing the univariate B-spline tri-product integrals

$$L_{\ell,i_\ell j_\ell k_\ell} = \int_0^1 \hat{\beta}_{\ell,i_\ell}\hat{\beta}_{\ell,j_\ell}\hat{\beta}_{\ell,k_\ell} \mathrm{d}\hat{x}_\ell, \tag{14}$$

we rewrite the sum (13) as

$$M_{\boldsymbol{ij}} \approx \sum_{\boldsymbol{k}\in\mathcal{I}} w_{\boldsymbol{k}} \prod_{\ell=1}^{3} L_{\ell,i_\ell j_\ell k_\ell}. \tag{15}$$

In the case of uniform knots, the majority of the tri-product integrals (14) (more precisely, all integrals that do not involve boundary B-splines) that are required for the matrix assembly can be derived from the standardized B-spline tri-product integrals (SBTI)

$$\Lambda_{jk}^{\delta\,\delta'\delta''} = \int_{\mathbb{R}} \beta^{p,\delta}_{[0,1,\ldots,p+1]}(\hat{x})\beta^{p,\delta'}_{[j,j+1,\ldots,j+p+1]}(\hat{x})\beta^{p,\delta''}_{[k,k+1,\ldots,k+p+1]}(\hat{x})\mathrm{d}\hat{x} \tag{16}$$

with $0 \le j,k \le p$ and $\delta\delta'\delta'' = 0, \delta, \delta', \delta'' \in \{0,1\}$, which can be precomputed, forming a look-up table with $7(p+1)^2$ entries. Note that the case $\delta = \delta' = \delta'' = 1$ is not considered.

Table 1: The values of SBTI for degree $p = 2$.

j,k	Λ^{000}_{jk}	Λ^{001}_{jk}	Λ^{010}_{jk}	Λ^{011}_{jk}	Λ^{100}_{jk}	Λ^{101}_{jk}	Λ^{110}_{jk}
$0,0$	$12/35$	0	0	$2/5$	0	$2/5$	$2/5$
$0,1$	$43/420$	$31/120$	$-31/240$	$-7/40$	$-31/240$	$-7/40$	$17/60$
$0,2$	$1/840$	$1/120$	$-1/240$	$-1/40$	$-1/240$	$-1/40$	$1/60$
$1,0$	$43/420$	$-31/240$	$31/120$	$-7/40$	$-31/240$	$17/60$	$-7/40$
$1,1$	$43/420$	$31/240$	$31/240$	$17/60$	$-31/120$	$-7/40$	$-7/40$
$1,2$	$1/168$	$7/240$	0	$1/120$	$-7/240$	$-7/60$	$1/120$
$2,0$	$1/840$	$-1/240$	$1/120$	$-1/40$	$-1/240$	$1/60$	$-1/40$
$2,1$	$1/168$	0	$7/240$	$1/120$	$-7/240$	$1/120$	$-7/60$
$2,2$	$1/840$	$1/240$	$1/240$	$1/60$	$-1/120$	$-1/40$	$-1/40$

Table 2: The values of SBTI for degree $p = 3$, scaled by $K = 181,440$.

j,k	$K\Lambda^{000}_{jk}$	$K\Lambda^{001}_{jk}$	$K\Lambda^{010}_{jk}$	$K\Lambda^{011}_{jk}$	$K\Lambda^{100}_{jk}$	$K\Lambda^{101}_{jk}$	$K\Lambda^{110}_{jk}$
$0,0$	$47,496$	0	0	$42,840$	0	$42,840$	$42,840$
$0,1$	$18,871$	$35,682$	$-17,841$	$-14,139$	$-17,841$	$-14,139$	$33,885$
$0,2$	868	$3,888$	$-1,944$	$-7,236$	$-1,944$	$-7,236$	$5,148$
$0,3$	1	10	-5	-45	-5	-45	27
$1,0$	$18,871$	$-17,841$	$35,682$	$-14,139$	$-17,841$	$33,885$	$-14,139$
$1,1$	$18,871$	$17,841$	$17,841$	$33,885$	$-35,682$	$-14,139$	$-14,139$
$1,2$	$2,550$	$8,130$	0	$2,646$	$-8,130$	$-21,546$	$2,646$
$1,3$	17	129	-21	-135	-108	-711	153
$2,0$	868	$-1,944$	$3,888$	$-7,236$	$-1,944$	$5,148$	$-7,236$
$2,1$	$2,550$	0	$8,130$	$2,646$	$-8,130$	$2,646$	$-21,546$
$2,2$	868	$1,944$	$1,944$	$5,148$	$-3,888$	$-7,236$	$-7,236$
$2,3$	17	108	21	153	-129	-711	-135
$3,0$	1	-5	10	-45	-5	27	-45
$3,1$	17	-21	129	-135	-108	153	-711
$3,2$	17	21	108	153	-129	-135	-711
$3,3$	1	5	5	27	-10	-45	-45

Here $\beta^{p,\delta}_{\Xi}(\hat{x})$ represents the derivative of order δ of the univariate B-spline with local knot vector Ξ. Table 1 and 2 present the look-up tables composed of the SBTI (16) for quadratic and cubic splines, respectively. Owing to the properties of B-splines, the SBTI values are related to the quantities (14) via

$$L_{\ell,i_\ell j_\ell k_\ell} = h_\ell \begin{cases} \Lambda^{000}_{j_\ell - i_\ell, k_\ell - i_\ell} & \text{if} \quad i_\ell = \min\{i_\ell, j_\ell, k_\ell\} \\ \Lambda^{000}_{i_\ell - j_\ell, k_\ell - j_\ell} & \text{if} \quad j_\ell = \min\{i_\ell, j_\ell, k_\ell\} \\ \Lambda^{000}_{i_\ell - k_\ell, j_\ell - k_\ell} & \text{if} \quad k_\ell = \min\{i_\ell, j_\ell, k_\ell\} \end{cases}, \tag{17}$$

where h_ℓ is the mesh size of B-splines $\hat{\beta}_{\ell,i_\ell}, \hat{\beta}_{\ell,j_\ell}, \hat{\beta}_{\ell,k_\ell}$. Only SBTI values with $\delta = \delta' = \delta'' = 0$ are needed for assembling the mass matrix according to (15), while assembling the stiffness matrix requires all entries of the look-up table.

Remark 2. This approach is also suitable for non-uniform B-splines. In this situation,

we generate look-up tables for all the tri-product integrals (14) in each direction via Gauss quadrature. Unlike the uniform case, the size of the look-up table for non-uniform B-splines not only depends on the spline degree, but also the number of basis functions in each direction. Using the quantities introduced in Appendix A, we can estimate the number of entries in this look-up table as

$$7 \sum_{\ell=1}^{3} \sum_{i_\ell=1}^{n} \sum_{j_\ell=\max\{1,i_\ell-p\}}^{\min\{n,i_\ell+p\}} C_3(i_\ell, j_\ell).$$

The number of quadrature nodes needed to evaluate one of the entries tends to

$$\lceil \frac{3p+1}{2} \rceil \bar{E}_3 = \mathcal{O}(p^2)$$

as n increases, where $\bar{E}_3$ is the average number of elements in the support intersection of three B-splines, provided that this intersection is non-empty. The precise number is not of much interest, since the total number of quadrature nodes per degree of freedom (and hence the number of required floating point operations per degree of freedom) tends to

$$\begin{aligned} \lim_{n\to\infty} \frac{1}{n^3} \Big(7 \sum_{\ell=1}^{3} \sum_{i_\ell=1}^{n} \sum_{j_\ell=\max\{1,i_\ell-p\}}^{\min\{n,i_\ell+p\}} C_3(i_\ell, j_\ell)\Big) \Big(\lceil \frac{3p+1}{2} \rceil \bar{E}_3\Big) \\ = \lim_{n\to\infty} \frac{1}{n^3} \left(21 n \bar{C}_2 \bar{C}_3\right) \Big(\lceil \frac{3p+1}{2} \rceil \bar{E}_3\Big) = 0. \end{aligned}$$

We conclude that the generation of these look-up tables does not contribute significantly to the overall computational costs. ◇

6. Step III: Evaluation via sum factorization

The interpolation and look-up (IL) method established in [32] performs the matrix assembly via the spline projection and look-up as discussed above, resulting in a complexity of $\mathcal{O}(n^3 p^6)$. Taking the IL approach as a starting point, we further reduce the computational cost of matrix assembly by exploiting the tensor-product structure via the sum factorization technique. This technique is widely applied in spectral methods [36], and has also been employed for efficient assembling of matrices in isogeometric analysis [3, 12, 15, 23, 40].

We derive the interpolation, look-up and sum-factorization (ILS) method by rewriting the sum in (15) as

$$M_{\boldsymbol{ij}} \approx \underbrace{\sum_{k_3} L_{3,i_3 j_3 k_3} \underbrace{\sum_{k_2} L_{2,i_2 j_2 k_2} \underbrace{\sum_{k_1} L_{1,i_1 j_1 k_1} w_{k_1 k_2 k_3}}_{= M^{\mathrm{ILS},1}_{i_1 j_1 k_2 k_3}}}_{= M^{\mathrm{ILS},2}_{i_1 i_2 j_1 j_2 k_3}}}_{= M^{\mathrm{ILS},3}_{i_1 i_2 i_3 j_1 j_2 j_3}}, \tag{18}$$

where we introduce three auxiliary tensors $\boldsymbol{M}^{\mathrm{ILS},1}$, $\boldsymbol{M}^{\mathrm{ILS},2}$ and $\boldsymbol{M}^{\mathrm{ILS},3}$. Subsequently we calculate the sums for all pairs $(\boldsymbol{i},\boldsymbol{j})$ via the following algorithm.

Algorithm MassMatrixAssembly ▷ ILS algorithm
for i_1 **from** 1 **to** n **do**
  for j_1 **from** $\max(1, i_1 - p)$ **to** $\min(n, i_1 + p)$ **do** ▷ supports of $\hat{\beta}_{1,i_1}$ and $\hat{\beta}_{1,j_1}$ intersect
    for k_2 **from** 1 **to** n **do**
      for k_3 **from** 1 **to** n **do**
        $M^{\mathrm{ILS},1}_{i_1 j_1 k_2 k_3} = 0$ ▷ initialization of 1st tensor
        for k_1 **from** $\max(1, i_1 - p, j_1 - p)$ **to** $\min(n, i_1 + p, j_1 + p)$ **do**
          ▷ supports of $\hat{\beta}_{1,i_1}$, $\hat{\beta}_{1,j_1}$ and $\hat{\beta}_{1,k_1}$ intersect
          $M^{\mathrm{ILS},1}_{i_1 j_1 k_2 k_3} \mathrel{+}= L_{1,i_1 j_1 k_1} w_{k_1 k_2 k_3}$ ▷ evaluation of 1st tensor
  for i_2 **from** 1 **to** n **do**
    for j_2 **from** $\max(1, i_2 - p)$ **to** $\min(n, i_2 + p)$ **do** ▷ supports of $\hat{\beta}_{2,i_2}$ and $\hat{\beta}_{2,j_2}$ intersect
      for k_3 **from** 1 **to** n **do**
        $M^{\mathrm{ILS},2}_{i_1 i_2 j_1 j_2 k_3} = 0$ ▷ initialization of 2nd tensor
        for k_2 **from** $\max(1, i_2 - p, j_2 - p)$ **to** $\min(n, i_2 + p, j_2 + p)$ **do**
          ▷ supports of $\hat{\beta}_{2,i_2}$, $\hat{\beta}_{2,j_2}$ and $\hat{\beta}_{2,k_2}$ intersect
          $M^{\mathrm{ILS},2}_{i_1 i_2 j_1 j_2 k_3} \mathrel{+}= L_{2,i_2 j_2 k_2} M^{\mathrm{ILS},1}_{i_1 j_1 k_2 k_3}$ ▷ evaluation of 2nd tensor
      for i_3 **from** 1 **to** n **do**
        for j_3 **from** $\max(1, i_3 - p)$ **to** $\min(n, i_3 + p)$ **do**
          ▷ supports of $\hat{\beta}_{3,i_3}$ and $\hat{\beta}_{3,j_3}$ intersect
          $M^{\mathrm{ILS},3}_{i_1 i_2 i_3 j_1 j_2 j_3} = 0$ ▷ initialization of 3rd tensor
          for k_3 **from** $\max(1, i_3 - p, j_3 - p)$ **to** $\min(n, i_3 + p, j_3 + p)$ **do**
            ▷ supports of $\hat{\beta}_{3,i_3}$, $\hat{\beta}_{3,j_3}$ and $\hat{\beta}_{3,k_3}$ intersect
            $M^{\mathrm{ILS},3}_{i_1 i_2 i_3 j_1 j_2 j_3} \mathrel{+}= L_{3,i_3 j_3 k_3} M^{\mathrm{ILS},2}_{i_1 i_2 j_1 j_2 k_3}$ ▷ evaluation of 3rd tensor
return $\boldsymbol{M}^{\mathrm{ILS},3}$

7. Computational costs and comparison with other methods

We analyze the computational complexity of our method by calculating the asymptotic number of floating point operations (flops) per degree of freedom (dof). More precisely, the following quantity

$$A = \lim_{n\to\infty} \frac{T}{n^3} \tag{19}$$

will be evaluated, where T denotes the total number of flops.

The proposed ILS method, which is summarized in Section 3, consists of three steps: Projecting the common factor in the required integrals into a tensor-product spline space, building a look-up table and assembling the matrix.

For the spline projection step, when using quasi-interpolation operator, note that there are no more than $p+2$ non-zero coefficients $\alpha_{\ell,k_\ell j_\ell}$ for each instance of k_ℓ in (12). The tensor-product structure indicates that sum factorization can be used to perform this computation,

which yields

$$w_{\boldsymbol{k}} = \sum_{j_3} \alpha_{3,k_3 j_3} \underbrace{\sum_{j_2} \alpha_{2,k_2 j_2} \underbrace{\sum_{j_1} \alpha_{1,k_1 j_1} w(\gamma_{1,j_1}, \gamma_{2,j_2}, \gamma_{3,j_3})}_{= w^{(1)}_{k_1 j_2 j_3}}}_{= w^{(2)}_{k_1 k_2 j_3}}}_{= w^{(3)}_{k_1 k_2 k_3}}, \tag{20}$$

where three auxiliary tensors $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$ and $\boldsymbol{W}^{(3)}$ are introduced. Indeed the sum-factorization technique has also been used in global surface interpolation [37, Section 9.2.5]. Similar to Algorithm MassMatrixAssembly, we perform the summations (20) in a recursive way. It suffices to consider the non-zero elements and terms in the tensors.

The first (innermost) loop evaluates about $n \cdot n \cdot n$ non-zero elements $w^{(1)}_{k_1 j_2 j_3}$ by summing over at most $p+2$ indices j_1 and requires 2 flops per term. The evaluation of the second auxiliary tensor with about $n \cdot n \cdot n$ non-zero elements $w^{(2)}_{k_1 k_2 j_3}$ proceeds by summing over at most $p+2$ indices j_2 and needs 2 flops per term. Finally we evaluate the $n \cdot n \cdot n$ non-zero elements $w^{(3)}_{k_1 k_2 k_3}$ by summing over at most $p+2$ indices j_3, requiring again 2 flops per term. Summing up, the quasi-interpolation operator requires not more than $6n^3(p+2)$ flops and thus asymptotically not more than $6(p+2)$ flops per dof.

Another approach to the spline projection proceeds simply by interpolation at the Greville abscissas. As analyzed in [32], this approach has a similar computational complexity as quasi-interpolation.

We already noted that the size of the look-up table for uniform and non-uniform univariate B-spline tri-product integrals is $\mathcal{O}(p^2)$ and $\mathcal{O}(np^2)$ respectively, and the computational effort required to build it does not contribute significantly to the overall computational effort.

As stated in Algorithm MassMatrixAssembly, the task is to recursively evaluate the nonzero elements of three auxiliary tensors. In order to facilitate the analysis of the computational complexity of this step, we introduce some quantities which are presented in Appendix A. The first loop of the algorithm evaluates

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot n \cdot n$$

non-zero elements $M^{\mathrm{ILS},1}_{i_1 j_1 k_2 k_3}$ by summing over $C_3(i_1, j_1)$ indices k_1 for each pair (i_1, j_1) and needs 2 flops per term. The second loop computes

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot n$$

non-zero elements $M^{\mathrm{ILS},2}_{i_1 i_2 j_1 j_2 k_3}$ by summing over $C_3(i_2, j_2)$ indices k_2 for each pair (i_2, j_2) and

needs 2 flops per term. Finally we evaluate

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot \sum_{i_3=1}^{n} C_2(i_3)$$

non-zero elements $M^{\mathrm{ILS},3}_{i_1 i_2 i_3 j_1 j_2 j_3}$ by summing over $C_3(i_3, j_3)$ indices k_3 for each pair (i_3, j_3), performing 2 flops per term. Therefore, as n increases, the number of flops per dof required by the matrix assembly step tends to $2\bar{C}_2\bar{C}_3(1 + \bar{C}_2 + \bar{C}_2^2)$.

Summing up, the asymptotic number of flops per dof for the ILS method does not exceed

$$6(p+2) + 2\bar{C}_2\bar{C}_3(1 + \bar{C}_2 + \bar{C}_2^2) = 24p^4 + 60p^3 + 62p^2 + 36p + 18. \tag{21}$$

We compare the ILS method with several other approaches, including Gauss quadrature (GQ), element-wise Gauss quadrature with sum factorization (EGS) [3], interpolation and look-up (IL) [32], weighted quadrature (WQ) [15] and global Gauss quadrature with sum factorization (GGS) [12]. Appendix C provides a detailed description of these previous approaches.

Remark 3. We notice that the tensors $\boldsymbol{M}^{\mathrm{ILS},\ell}$ ($\ell = 1, 2, 3$) introduced in Algorithm MassMatrixAssembly possess up to three symmetries. Indeed, it is always possible to swap the indices i_ℓ and j_ℓ, $\ell = 1, 2, 3$, without altering the value of the tensor element. Consequently, the mass matrix assembly via ILS can be performed even more efficiently by exploiting these symmetries. The resulting method will be denoted as ILS-S. A detailed analysis (not presented here) shows that one achieves an asymptotic number of flops per dof not exceeding

$$3p^4 + 14p^3 + 26p^2 + 27p + 18. \tag{22}$$

The leading coefficient is reduced by the factor $1/2^3 = 1/8$, due to the three symmetries that are present in the final matrix. In fact, the third step of the assembly – where the highest number of symmetries is present – dominates the complexity with respect to the degree p. A similar speedup can be achieved for all the other approaches that preserve the symmetry of the mass matrix (GQ, EGS, IL and GGS). ◇

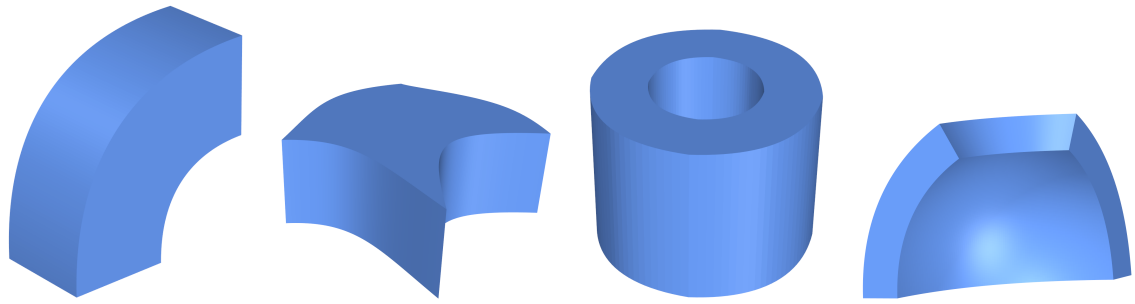
Table 3 lists the asymptotic number of flops per dof for $p = 1, \dots 5$ as n increases, and the asymptotics of that number as p increases. The ILS method has a significant advantage over the other methods that also preserve the symmetry of the resulting matrix. This is not the case for WQ, which achieves the same complexity with respect to the degree p and even leads to a smaller value of the leading coefficient (approx. 66.7% of ILS). All methods except WQ can be further accelerated by exploiting the symmetries to reduce the computational effort. The resulting ILS-S method possesses the same complexity as ILS, and achieves both the smallest number of flops per dof for all degrees and a smaller value of the leading coefficient (12.5% of ILS and 18.75% of WQ).

Table 3: Asymptotic number of floating point operations per degree of freedom for different matrix assembly approaches. Note that the numbers for ILS and ILS-S are an upper bound.

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$	asymptotics
GQ	1,672	60,534	794,688	5,890,750	30,326,616	$\mathcal{O}(p^9)$
EGS	512	7,047	47,104	209,375	715,392	$\mathcal{O}(p^7)$
IL	1,390	27,460	202,642	907,960	3,014,326	$\mathcal{O}(p^6)$
GGS	304	2,646	11,904	37,750	96,336	$\mathcal{O}(p^5)$
WQ	190	1,014	3,350	8,446	17,934	$16p^4 + \mathcal{O}(p^3)$
ILS	200	1,202	4,248	11,138	24,248	$24p^4 + \mathcal{O}(p^3)$
ILS-S	88	336	954	2,206	4,428	$3p^4 + \mathcal{O}(p^3)$

8. Numerical experiments

We test the performance of the ILS method on four computational domains, taken from the `G+Smo` test suite [20], focusing on computational complexity and numerical behaviour. Figure 2 visualizes these domains and provides information regarding the number of control points and the degrees of the parameterizations. We also compare the computation times with Gauss quadrature (GQ) and with the interpolation and look-up (IL) method. All of these approaches are implemented in C++ using the `G+Smo` library [29], and all the experiments are conducted on a laptop computer of an Intel Core i5-7300HQ CPU, 2.5 GHz with 8 GB memory.



(a) 2×3×2, $\boldsymbol{p}$=(1, 2, 1) (b) 5×5×2, $\boldsymbol{p}$=(4, 4, 1) (c) 9×2×5, $\boldsymbol{p}$=(2, 1, 2) (d) 3×3×2, $\boldsymbol{p}$=(2, 2, 1)

Figure 2: Computational domains used for testing the performance of different methods.

8.1. Numerical behaviour

In order to evaluate numerically the behaviour of the proposed method, which includes the computational complexity with respect to the number of degrees of freedom n^3 and the spline degree p, we consider a 3D Poisson problem on each of the four computational domains shown in Figure 2.

First we study the dependence of the time needed for generating the matrices on the number of degrees of freedom n^3. Figures 3(a)-3(d) present the observed time required for assembling the system matrix with various p and n^3 on four different domains, respectively.

The computational time depends linearly on the number of degrees of freedom, which is in line with the expected results.

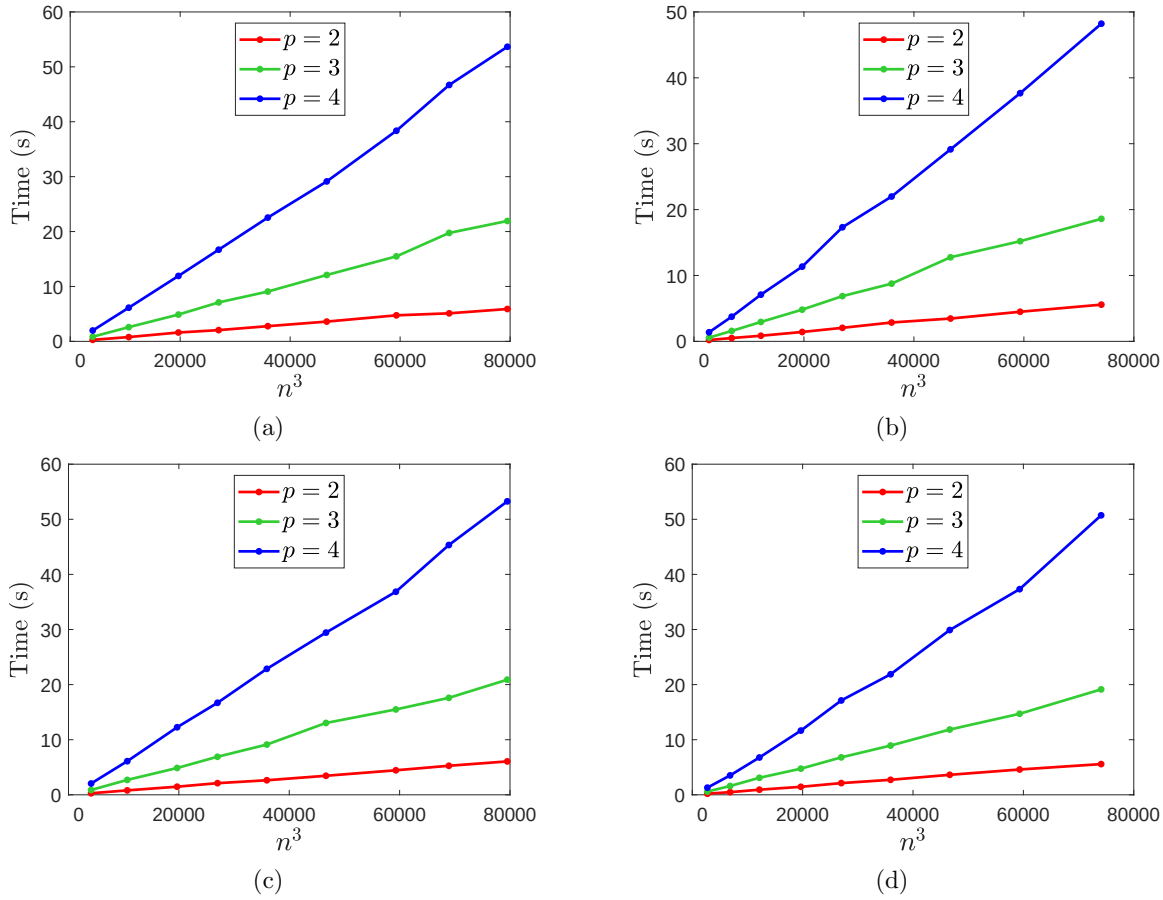
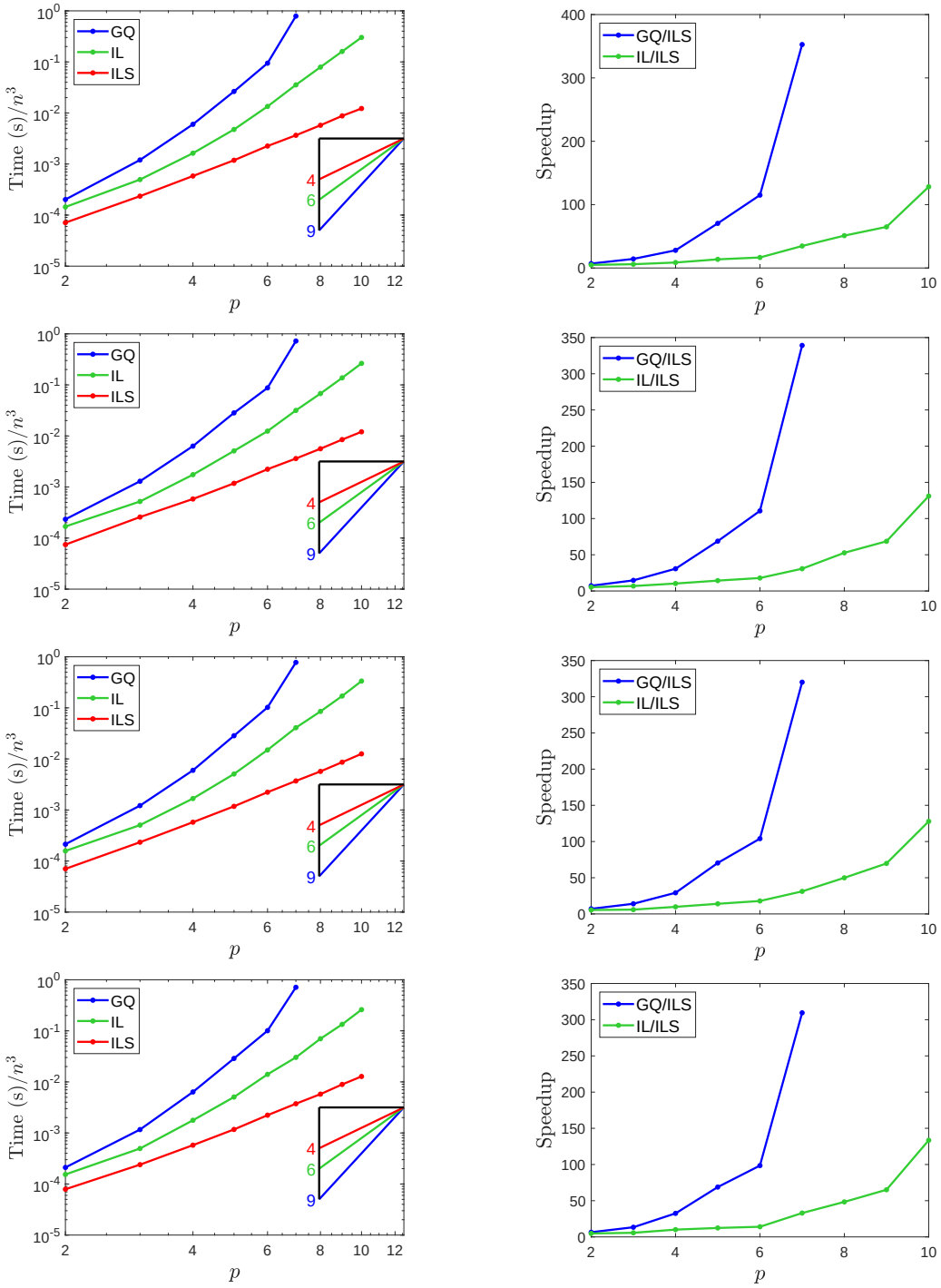


Figure 3: Computational time of our ILS method for assembling the system matrix on different domains with various spline degrees and numbers of degrees of freedom.

Next we investigate the relation between the computational time and the degree p for different methods. In order to eliminate the effect of the number of degrees of freedom on the result, we divide the computational time by n^3 in all examples.

Figure 4(a) shows a log-log plot of the dependence of the assembling time by different methods and the spline degree. Here we consider relatively large degrees p (up to 10) and estimate the slopes to evaluate the asymptotic behaviour. For Gauss quadrature (GQ), we use $p+1$ nodes per element in each direction, which is proved to guarantee optimality of the Galerkin formulation with numerical quadrature [32]. All the plots confirm the theoretical asymptotic behaviour of IL and ILS with respect to the degree p, which were analyzed in Section 7. In addition, Figure 4(b) reports the achieved speedup with respect to GQ and IL.



(a) Degree-dependence of assembly time (b) Speedup

Figure 4: (a) Computational time for assembling the system matrix with various spline degrees ($p = 2, \ldots, 10$). The computational time of Gauss quadrature for $p > 7$ is not included. (b) Visualization of the observed speedup with respect to GQ and IL. Each row corresponds to one computational domain.

8.2. Computational time comparisons

In addition, we summarize the computational time of the different methods for various spline degrees and computational domains in Table 4. The corresponding speedup factors between our method and the others are also reported. For Gauss quadrature (GQ), we limit the degree to six since the computation for higher degree is too time-consuming (more than 2 hours). Moreover, we use the same number of knot spans (24 in our tests) in all directions and for all degrees, for simplicity. Note that the discretization is not isogeometric if the degree of the parameterization exceeds p, which happens in some of these cases.

Clearly, our method is significantly faster than other state-of-the-art matrix-assembly approaches, especially for high spline degrees. Nevertheless, due to the common use of low degrees in isogeometric applications, the speedup achieved for these degrees is equally important, and the ILS approach is beneficial also in this situation.

The speedup factors do not match the theoretically predicted values as noted in Table 5. Indeed, for $p \geq 3$, the ILS is method is approximately 18 times slower than predicted theoretically. This is due to implementation aspects, since the development team of the `G+Smo` library [29] invested substantial efforts and manpower into the efficient implementation of the matrix assembly via Gaussian quadrature. Nevertheless, these numbers still indicate the advantages of the new method.

9. Matrix-free applications using ILS

In addition to assembling the stiffness and mass matrix, another important step in solving PDEs using IgA is the numerical solution of the assembled linear system (5), which often relies on iterative methods. These methods only require the ability to evaluate matrix-vector products, while a direct access to the matrix elements is not mandatory. This observation has been exploited in recent publications [12, 40], which propose a matrix-free strategy, merging the assembly and linear system solving into a single operator application step. This avoids storing the stiffness and mass matrix explicitly.

9.1. Algorithm for matrix-vector products using ILS

For the simplicity of presentation, we only consider the products of the mass matrix M and a vector $\boldsymbol{u}$. Based on (18) we arrive at

$$
\begin{aligned}
(M\boldsymbol{u})_{\boldsymbol{i}} &= \sum_{\boldsymbol{j}} M_{\boldsymbol{ij}} u_{\boldsymbol{j}} \qquad (23)\\
&\approx \sum_{j_3}\sum_{k_3} L_{3,i_3j_3k_3} \underbrace{\sum_{j_2}\sum_{k_2} L_{2,i_2j_2k_2} \underbrace{\sum_{j_1} u_{j_1j_2j_3} \underbrace{\sum_{k_1} w_{k_1k_2k_3} L_{1,i_1j_1k_1}}_{=V^{(1)}_{i_1j_1k_2k_3}}}_{=V^{(2)}_{i_1j_2j_3k_2k_3}}}_{=V^{(3)}_{i_1i_2j_3k_3}}, \qquad \boldsymbol{i} \in \mathcal{I},
\end{aligned}
$$

Table 4: Assembly time and speedup for various spline degrees, numbers of degrees of freedom and computational domains by different methods.

Domain	Degree	#DOF	Assembly time (s)			Speedup	
			GQ	IL	ILS	GQ/ILS	IL/ILS
(a)	1	15,625	5.91e-1	5.71e-1	1.09e-1	5.42e0	5.24e0
	2	17,576	3.52e0	2.52e0	5.34e-1	6.59e0	4.71e0
	3	19,683	2.36e1	9.76e0	1.74e0	1.36e1	5.61e0
	4	21,952	1.33e2	3.61e1	4.45e0	2.99e1	8.13e0
	5	24,389	6.84e2	1.23e2	8.98e0	7.62e1	1.37e1
	6	27,000	2.55e3	3.62e2	2.31e1	1.11e2	1.57e1
	7	29,791	n/a	1.06e3	3.34e1	n/a	3.17e1
	8	32,768	n/a	2.59e3	5.27e1	n/a	4.92e1
	9	35,937	n/a	5.76e3	9.21e1	n/a	6.25e1
(b)	1	15,625	8.39e-1	8.29e-1	1.08e-1	7.77e0	7.68e0
	2	17,576	4.04e0	2.92e0	5.24e-1	7.70e0	5.57e0
	3	19,683	2.48e1	9.95e0	1.65e0	1.51e1	6.05e0
	4	21,952	1.44e2	3.97e1	4.15e0	3.46e1	9.55e0
	5	24,389	6.59e2	1.19e2	8.95e0	7.37e1	1.32e1
	6	27,000	2.75e3	3.90e2	2.30e1	1.18e2	1.69e1
	7	29,791	n/a	9.92e2	3.42e1	n/a	2.90e1
	8	32,768	n/a	2.42e3	5.02e1	n/a	4.82e1
	9	35,937	n/a	5.53e3	8.32e1	n/a	6.65e1
(c)	1	15,625	6.41e-1	6.25e-1	1.11e-1	5.78e0	5.63e0
	2	17,576	3.68e0	2.88e0	5.33e-1	6.90e0	5.40e0
	3	19,683	2.34e1	9.73e0	1.72e0	1.36e1	5.67e0
	4	21,952	1.39e2	3.90e1	4.21e0	3.32e1	9.28e0
	5	24,839	6.94e2	1.24e2	9.13e0	7.60e1	1.35e1
	6	27,000	2.76e3	4.04e2	2.35e1	1.18e2	1.72e1
	7	29,791	n/a	1.02e3	2.98e1	n/a	3.41e1
	8	32,768	n/a	2.59e3	5.00e1	n/a	5.18e1
	9	35,937	n/a	5.83e3	9.11e1	n/a	6.40e1
(d)	1	15,625	5.97e-1	6.23e-1	1.10e-1	5.43e0	5.66e0
	2	17,576	3.47e0	2.53e0	5.43e-1	6.38e0	4.65e0
	3	19,683	2.17e1	9.23e0	1.76e0	1.24e1	5.26e0
	4	21,952	1.49e2	4.15e1	4.31e0	3.45e1	9.61e0
	5	24,389	6.71e2	1.18e2	1.01e1	6.63e1	1.16e1
	6	27,000	2.43e3	3.39e2	2.61e1	9.32e1	1.30e1
	7	29,791	n/a	1.00e3	3.19e1	n/a	3.13e1
	8	32,768	n/a	2.40e3	5.34e1	n/a	4.48e1
	9	35,937	n/a	5.71e3	9.21e1	n/a	6.20e1

Table 5: The predicted and experimentally observed speedup of the ILS method with respect to Gauss quadrature. For each degree, we take the corresponding minimal speedup value on four domains listed in Table 4 as the observed speedup value, and the predicted value is obtained from Table 3 directly.

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
Predicted speedup	8.36	50.36	187.07	528.89	1250.69
Observed speedup	5.42	6.38	12.4	29.9	66.3

where three auxiliary tensors $\boldsymbol{V}^{(1)}$, $\boldsymbol{V}^{(2)}$ and $\boldsymbol{V}^{(3)}$ are introduced. Similar to Algorithm MassMatrixAssembly, the calculation of the vector $\boldsymbol{v} = M\boldsymbol{u}$ can also be performed in a recursive way, which is summarized in the following algorithm.

Algorithm MassMatrixVectorProduct
for i_1 **from** 1 **to** n **do**
  for k_2 **from** 1 **to** n **do**
    for k_3 **from** 1 **to** n **do**
      for j_1 **from** $\max(1, i_1 - p)$ **to** $\min(n, i_1 + p)$ **do** ▷ supports of $\hat{\beta}_{1,i_1}$ and $\hat{\beta}_{1,j_1}$ intersect
        $V^{(1)}_{i_1 j_1 k_2 k_3} = 0$ ▷ initialization of 1st tensor
        for k_1 **from** $\max(1, i_1 - p, j_1 - p)$ **to** $\min(n, i_1 + p, j_1 + p)$ **do**
          ▷ supports of $\hat{\beta}_{1,i_1}$, $\hat{\beta}_{1,j_1}$ and $\hat{\beta}_{1,k_1}$ intersect
          $V^{(1)}_{i_1 j_1 k_2 k_3} \mathrel{+}= w_{k_1 k_2 k_3} L_{1,i_1 j_1 k_1}$
      for j_2 **from** $\max(1, k_2 - p)$ **to** $\min(n, k_2 + p)$ **do** ▷ supports of $\hat{\beta}_{2,j_2}$ and $\hat{\beta}_{2,k_2}$ intersect
        for j_3 **from** $\max(1, k_3 - p)$ **to** $\min(n, k_3 + p)$ **do**
          ▷ supports of $\hat{\beta}_{3,j_3}$ and $\hat{\beta}_{3,k_3}$ intersect
          $V^{(2)}_{i_1 j_2 j_3 k_2 k_3} = 0$ ▷ initialization of 2nd tensor
          for j_1 **from** $\max(1, i_1 - p)$ **to** $\min(n, i_1 + p)$ **do**
            ▷ supports of $\hat{\beta}_{1,i_1}$ and $\hat{\beta}_{1,j_1}$ intersect
            $V^{(2)}_{i_1 j_2 j_3 k_2 k_3} \mathrel{+}= u_{j_1 j_2 j_3} V^{(1)}_{i_1 j_1 k_2 k_3}$
  for i_2 **from** 1 **to** n **do**
    for j_3 **from** 1 **to** n **do**
      for k_3 **from** $\max(1, j_3 - p)$ **to** $\min(n, j_3 + p)$ **do** ▷ supports of $\hat{\beta}_{3,j_3}$ and $\hat{\beta}_{3,k_3}$ intersect
        $V^{(3)}_{i_1 i_2 j_3 k_3} = 0$ ▷ initialization of 3rd tensor
        for j_2 **from** $\max(1, i_2 - p)$ **to** $\min(n, i_2 + p)$ **do**
          ▷ supports of $\hat{\beta}_{2,i_2}$ and $\hat{\beta}_{2,j_2}$ intersect
          for k_2 **from** $\max(1, i_2 - p, j_2 - p)$ **to** $\min(n, i_2 + p, j_2 + p)$ **do**
            ▷ supports of $\hat{\beta}_{2,i_2}$, $\hat{\beta}_{2,j_2}$ and $\hat{\beta}_{2,k_2}$ intersect
            $V^{(3)}_{i_1 i_2 j_3 k_3} \mathrel{+}= L_{2,i_2 j_2 k_2} V^{(2)}_{i_1 j_2 j_3 k_2 k_3}$
    for i_3 **from** 1 **to** n **do**
      $v_{i_1 i_2 i_3} = 0$
      for j_3 **from** $\max(1, i_3 - p)$ **to** $\min(n, i_3 + p)$ **do** ▷ supports of $\hat{\beta}_{3,i_3}$ and $\hat{\beta}_{3,j_3}$ intersect
        for k_3 **from** $\max(1, i_3 - p, j_3 - p)$ **to** $\min(n, i_3 + p, j_3 + p)$ **do**
          ▷ supports of $\hat{\beta}_{3,i_3}$, $\hat{\beta}_{3,j_3}$ and $\hat{\beta}_{3,k_3}$ intersect
          $v_{i_1 i_2 i_3} \mathrel{+}= L_{3,i_3 j_3 k_3} V^{(3)}_{i_1 i_2 j_3 k_3}$
return $\boldsymbol{v}$

9.2. Computational costs and comparison with other methods

To estimate the computational costs of Algorithm MassMatrixVectorProduct, it suffices to consider the non-zero tensor elements and terms in (23). The first step evaluates

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot n \cdot n$$

Table 6: Asymptotic number of flops per dof with respect to the matrix-free applications.

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$	asymptotics
WQ	96	160	224	288	352	$\mathcal{O}(p)$
GGS	128	549	1,600	3,725	7,488	$\mathcal{O}(p^4)$
ILS	124	516	1,352	2,800	5,028	$\mathcal{O}(p^3)$

non-zero elements $V^{(1)}_{i_1 j_1 k_2 k_3}$ by summing over $C_3(i_1, j_1)$ indices k_1 for each pair (i_1, j_1) and requires 2 flops per term. The evaluation of the second auxiliary tensor with

$$n \cdot \sum_{j_2=1}^{n} C_2(j_2) \cdot \sum_{j_3=1}^{n} C_2(j_3)$$

non-zero elements $V^{(2)}_{i_1 j_2 j_3 k_2 k_3}$ proceeds by summing over $C_2(i_1)$ indices j_1 for each i_1 and needs 2 flops per term. The third step, which computes the tensor $\boldsymbol{V}^{(3)}$ with

$$n \cdot n \cdot \sum_{j_3=1}^{n} C_2(j_3)$$

non-zero elements $V^{(3)}_{i_1 i_2 j_3 k_3}$ by summing over $C_2(i_2) \cdot C_3(i_2, j_2)$ indices for each i_2, (i_2, j_2) and requires 2 flops per term. The last step evaluates the $n \cdot n \cdot n$ non-zero elements $v_{\boldsymbol{i}}$ by summing over $C_2(i_3) \cdot C_3(i_3, j_3)$ indices for each i_3 and (i_3, j_3), requiring another 2 flops per term.

Summing up, the number of flops per dof tends to

$$2\bar{C}_2(\bar{C}_2^2 + \bar{C}_2\bar{C}_3 + 2\bar{C}_3)$$

as n increases, which brings the asymptotic number of flops per dof to

$$28p^3 + 54p^2 + 34p + 8 = \mathcal{O}(p^3).$$

We compare the ILS method with two other quadrature methods (WQ and GGS) in terms of matrix-free applications. The algorithms for the evaluation of matrix-vector products using these two techniques are described in Appendix D. Table 6 presents the asymptotic numbers of flops per dof for these matrix-free approaches. It can be seen that the computational cost of the GGS method is more expensive than our method, while the WQ method is even faster. However, it should be noted that the WQ method does not preserve the symmetry of the matrices and thus imposes constraints on the selection of suitable iterative solvers, cf. [44].

10. Assembling the stiffness matrix

Applying a spline projection (e.g., an interpolation or quasi-interpolation operator) defined in the space $\mathbb{S}$ to the common factor $\widetilde{D}$ in the occurring integrals (6), we obtain

$$\widetilde{D}(\hat{\boldsymbol{x}}) \approx \sum_{\boldsymbol{k} \in \mathcal{I}} \widetilde{D}_{\boldsymbol{k}} \hat{\beta}_{\boldsymbol{k}}(\hat{\boldsymbol{x}})$$

with the coefficient matrices $\widetilde{D}_{\boldsymbol{k}} = (d^{rs}_{\boldsymbol{k}})_{r,s=1,2,3}$. Consequently, the stiffness matrix is rewritten as

$$S_{\boldsymbol{ij}} = \int_{\hat{\Omega}} \hat{\nabla}\hat{\beta}_{\boldsymbol{i}}^T \widetilde{D} \hat{\nabla}\hat{\beta}_{\boldsymbol{j}} \mathrm{d}\hat{\boldsymbol{x}} \approx \sum_{r=1}^{3}\sum_{s=1}^{3} \underbrace{\sum_{\boldsymbol{k}} d^{rs}_{\boldsymbol{k}} \int_{\hat{\Omega}} (\hat{\partial}_r\hat{\beta}_{\boldsymbol{i}})(\hat{\partial}_s\hat{\beta}_{\boldsymbol{j}})\hat{\beta}_{\boldsymbol{k}} \mathrm{d}\hat{\boldsymbol{x}}}_{= S^{rs}_{\boldsymbol{ij}}}, \tag{24}$$

where the partial derivatives are defined as $\hat{\partial}_r\hat{\beta}_{\boldsymbol{i}} = \partial_r\hat{\beta}_{\boldsymbol{i}}/\partial\hat{x}_r$ and nine auxiliary tensors $\boldsymbol{S}^{rs}$ $(r,s=1,2,3)$ are introduced. For assembling the stiffness matrix using ILS, we need to apply this technique to each of the nine tensors. The computational cost of evaluating each tensor is the same as the one required for evaluating the mass matrix (7), and summing the nine tensors up requires

$$9 \cdot \sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot \sum_{i_3=1}^{n} C_2(i_3)$$

flops. Thus, the asymptotic number of flops per dof needed to assemble the stiffness matrix by ILS does not exceed

$$9\big(6(p+2) + 2\bar{C}_2\bar{C}_3(1+\bar{C}_2+\bar{C}_2^2)\big) + 9\bar{C}_2^3 = 9(24p^4+68p^3+74p^2+42p+19).$$

Similar to the case of mass matrix, the use of symmetries provides a further speedup. First, the tensors $\boldsymbol{S}^{\ell\ell}$ $(\ell = 1,2,3)$ possess three symmetries and can be evaluated at a reduced cost shown in (22). Second, the remaining six tensors $\boldsymbol{S}^{\ell\ell'}$ $(1 \le \ell,\ell' \le 3, \ell \ne \ell')$ can be organized into three pairs $S^{12}_{\boldsymbol{ij}} + S^{21}_{\boldsymbol{ij}}$, $S^{13}_{\boldsymbol{ij}} + S^{31}_{\boldsymbol{ij}}$ and $S^{23}_{\boldsymbol{ij}} + S^{32}_{\boldsymbol{ij}}$. The first pair can be rewritten as

$$S^{12}_{\boldsymbol{ij}} + S^{21}_{\boldsymbol{ij}} = \sum_{k_3} L_{3,i_3j_3k_3} \underbrace{\underbrace{\Big(\sum_{k_2} L_{2,i_2j_2'k_2} \underbrace{\sum_{k_1} d^{12}_{k_1k_2k_3} L_{1,i_1'j_1k_1}}_{= T^{(1)}_{i_1j_1k_2k_3}} + \sum_{k_2} L_{2,i_2'j_2k_2} \underbrace{\sum_{k_1} d^{21}_{k_1k_2k_3} L_{1,i_1j_1'k_1}}_{= T^{(2)}_{i_1j_1k_2k_3}}\Big)}_{= T^{(3)}_{i_1i_2j_1j_2k_3}}}_{= T^{(4)}_{i_1i_2i_3j_1j_2j_3}},$$

where we introduce four auxiliary tensors $\boldsymbol{T}^{(\ell)}$ $(\ell = 1,\ldots,4)$ and use look-up tables for integrals of derivatives (indicate by $'$) also. The tensor $T^{(4)}_{i_1i_2i_3j_1j_2j_3}$, whose assembly determines the coefficient of p^4 in the number of flops, is symmetric firstly with respect to swapping i_3 and j_3 and secondly with respect to swapping (i_1,i_2) and (j_1,j_2). The other two pairs can be dealt with in a similar way. A detailed analysis (not presented here) reveals that the asymptotic number of flops per dof for evaluating the stiffness matrix is reduced to

$$27p^4 + \mathcal{O}(p^3)$$

with the help of symmetries.

11. Summary and future work

Based on the techniques of (quasi-) interpolation, look-up and sum factorization, we developed an efficient algorithm for assembling the system matrices arising in isogeometric Galerkin discretizations. The resulting method requires $\mathcal{O}(Np^{d+1})$ flops for the matrix assembly task. It is competitive with weighted quadrature [15] in terms of the required number of flops, while preserving the symmetry of the matrices.

We also presented an analysis of the computational complexity for different approaches, which indicates that our method has a significant advantage over the other methods that preserve the symmetry of the matrix, both in terms of the asymptotic behaviour and for low polynomial degrees. Weighted quadrature, however, gives the same p-complexity of the number of flops per dof and lower costs (approx. 66%). Several numerical experiments confirmed these theoretical results. It was also shown that our method admits a further speedup (by a factor of approx. 8) based on the symmetries of the matrix.

The matrices assembled by the ILS method are exactly the same as the ones obtained with the IL approach. The theory developed in the work [32] shows that our method maintains the optimal rate of convergence, provided that the spline projection is sufficiently accurate.

In addition, the proposed new method can also be used for performing matrix-free applications. In this case, however, it is slower than weighted quadrature.

Regarding the future work, two further improvements and extensions will be considered. First, extending the proposed method to adaptively refined splines, such as HB-splines and THB-splines. Second, we plan to explore several options for optimizing the current code, including GPU computation [30], in order to close the gap between the predicted and observed speedup with respect to Gauss quadrature (see Table 5). The code of our implementation will be made available as part of the `G+Smo` C++ library for isogeometric analysis.

Acknowledgement

The authors gratefully acknowledge the support provided by the Austrian Science Fund (FWF) through the doctoral program W1214 "Computational Mathematics", and by the ERC Advanced Grant "CHANGE" (GA No. 694515). We also thank the reviewers for their comments, which were very helpful for preparing the revised version.

Appendix A. Overlapping functions and active elements

For analyzing the computational costs of matrix assembly approaches, we introduce the following quantities:

- First, the number $C_2(i_\ell)$ counts the B-splines that possess a non-empty support intersection with the fixed B-spline $\hat{\beta}_{\ell,i_\ell}$, where we only consider B-splines with single knots. As n increases, the average number tends to

$$\bar{C}_2 = \lim_{n\to\infty} \frac{1}{n} \sum_{i_\ell=1}^{n} C_2(i_\ell) = 2p+1.$$

- Second, the number $C_3(i_\ell, j_\ell)$ counts the B-splines that possess a non-empty support intersection with the product of two fixed B-splines $\hat{\beta}_{\ell,i_\ell}$ and $\hat{\beta}_{\ell,j_\ell}$, where we only consider pairs of B-splines with overlapping supports and single knots. The support intersection of B-splines $\hat{\beta}_{\ell,i_\ell}$ and $\hat{\beta}_{\ell,j_\ell}$ consists of at least 1 and at most $p+1$ elements. Consequently, there are at least $p+1$ and at most $2p+1$ B-splines $\hat{\beta}_{\ell,k_\ell}$ with a non-empty support intersection (i.e., $\hat{\beta}_{\ell,i_\ell}\hat{\beta}_{\ell,j_\ell}\hat{\beta}_{\ell,k_\ell} \neq 0$). Considering a fixed B-spline $\hat{\beta}_{\ell,i_\ell}$, we compute the average number $\bar{C}_3$, varying over the $2p+1$ different possibilities for choosing $\hat{\beta}_{\ell,j_\ell}$. As n increases, the average number tends to

$$\begin{aligned}\bar{C}_3 &= \lim_{n\to\infty} \frac{1}{\sum_{i_\ell=1}^{n} C_2(i_\ell)} \sum_{i_\ell=1}^{n} \sum_{j_\ell=\max\{1,i_\ell-p\}}^{\min\{n,i_\ell+p\}} C_3(i_\ell, j_\ell) \\ &= \frac{1}{2p+1}\big((p+1)+\ldots+2p+(2p+1)+2p+\ldots+(p+1)\big) = \frac{3p^2+3p+1}{2p+1}.\end{aligned}$$

- Third, the number $E_2(i_\ell, j_\ell)$ counts the elements in the support intersection of two B-splines $\hat{\beta}_{\ell,i_\ell}$ and $\hat{\beta}_{\ell,j_\ell}$. Again, we only consider pairs of B-splines with overlapping supports and single knots. The support intersection of these two B-splines consists of at least 1 and at most $p+1$ elements, which brings $2p+1$ different possibilities for choosing $\hat{\beta}_{\ell,j_\ell}$. As n increases, the average number $\bar{E}_2$ tends to

$$\begin{aligned}\bar{E}_2 &= \lim_{n\to\infty} \frac{1}{\sum_{i_\ell=1}^{n} C_2(i_\ell)} \sum_{i_\ell=1}^{n} \sum_{j_\ell=\max\{1,i_\ell-p\}}^{\min\{n,i_\ell+p\}} E_2(i_\ell, j_\ell) \\ &= \frac{1}{2p+1}(1+\ldots+p+(p+1)+p+\ldots+1) = \frac{(p+1)^2}{2p+1}.\end{aligned}$$

Appendix B. Quasi-interpolation for splines with non-uniform knots

We explain this technique in the *univariate* case, where we consider splines of degree p with knots

$$(\kappa_{-p}, \ldots, \kappa_0, \kappa_1, \kappa_2, \ldots, \kappa_{n-p-1}, \kappa_{n-p}, \ldots, \kappa_n) \tag{B.1}$$

that satisfy $\kappa_i \leq \kappa_{i+1}$ and $\kappa_i < \kappa_{i+p+1}$. In particular, it is standard to use the p-fold boundary knots $0 = \kappa_{1-p} = \ldots = \kappa_0$ and $\kappa_{n-p} = \ldots = \kappa_{n-1} = 1$. The two extremal knots κ_{-p} and κ_n are arbitrary. In fact, they do not have any influence on the B-splines' values on the domain $[\kappa_0, \kappa_{n-p}]$ except that they should be non-positive and not less than 1, respectively.

Recall that the n B-splines $\hat{\beta}_i$, $i = 1, \ldots, n$, have associated Greville abscissas

$$\gamma_i = \frac{1}{p} \sum_{j=i-p}^{i-1} \kappa_j.$$

We consider the $n-p$ sub-sequences $(\gamma_j, \ldots, \gamma_{j+p})$ with indices $j = 1, \ldots, n-p$ that contain $p+1$ consecutive Greville abscissas, and use interpolation to define associated polynomials

$$p_j(\hat{x}) = \sum_{k=j}^{j+p} \lambda_{j,k}(\varphi)\,(\hat{x} - \gamma_k)^p, \tag{B.2}$$

satisfying $p_j(\gamma_k) = \varphi(\gamma_k)$ for $k = j, \ldots, j+p$. Simple closed-form expressions for their coefficients $\lambda_{j,k}(\varphi)$ are available for low degrees p:

- $p = 2$, index triplets $(k, k', k'') = (j, j{+}1, j{+}2)$ and all permutations thereof:

$$\lambda_{j,k}(\varphi) = \frac{1}{2}\Big(\frac{\varphi(\gamma_{k'})}{(\gamma_k - \gamma_{k'})^2} + \frac{\varphi(\gamma_{k''})}{(\gamma_k - \gamma_{k''})^2} - \frac{(\gamma_{k'} - \gamma_{k''})^2\varphi(\gamma_k)}{(\gamma_k - \gamma_{k'})^2(\gamma_k - \gamma_{k''})^2}\Big) \tag{B.3}$$

- $p = 3$, index quadruplets $(k, k', k'', k''') = (j, j{+}1, j{+}2, j{+}3)$ and all permutations thereof:

$$\lambda_{j,k}(\varphi) = \frac{(\gamma_{k''} - \gamma_{k'''})^3\varphi(\gamma_{k'}) + (\gamma_{k'''} - \gamma_{k'})^3\varphi(\gamma_{k''}) + (\gamma_{k'} - \gamma_{k''})^3\varphi(\gamma_{k'''})}{3(\gamma_{k'} - \gamma_k)(\gamma_{k''} - \gamma_k)(\gamma_{k'''} - \gamma_k)(\gamma_{k'} - \gamma_{k''})(\gamma_{k''} - \gamma_{k'''})(\gamma_{k'''} - \gamma_{k'})} \tag{B.4}$$

The formulas for general degree p can be derived via the blossoms of the Lagrange basis polynomials. The particular representation (B.2) of these polynomials ensures the symmetry of the coefficient functionals.

The spline coefficients of the interpolating polynomials (B.2) are obtained via the blossoming approach,

$$\nu^p_{j,i}(\varphi) = \sum_{k=j}^{j+p} \lambda_{j,k}(\varphi) \prod_{\ell=1}^{p} (\kappa_{i-\ell} - \gamma_k). \tag{B.5}$$

In the case of *even* degree splines, the functionals μ^p_i appearing in the quasi-interpolation operator are directly determined by these coefficients. More precisely, we consider the polynomial that interpolates φ at the Greville abscissas centered at γ_i, and the boundary polynomials otherwise. The spline coefficients of this polynomial is used to compute the value of the functional,

$$\mu^p_i(\varphi) = \begin{cases} \nu^p_{1,i}(\varphi) & \text{if } 1 \le i < \frac{p}{2}, \\ \nu^p_{i-\frac{p}{2},i}(\varphi) & \text{if } 1 + \frac{p}{2} \le i \le n - \frac{p}{2}, \\ \nu^p_{n-p,i}(\varphi) & \text{if } n - \frac{p}{2} < i \le n. \end{cases} \tag{B.6}$$

In the case of *odd* degree splines, we proceed similarly but need to average the coefficients of two polynomials, in order to obtain symmetric formulas:

$$\mu^p_i(\varphi) = \begin{cases} \nu^p_{1,i}(\varphi) & \text{if } 1 \le i < \frac{p+1}{2}, \\ \dfrac{\gamma_i - \kappa_{i-p-1}}{\kappa_i - \kappa_{i-p-1}}\nu^p_{i-\frac{p+1}{2},i}(\varphi) + \dfrac{\kappa_i - \gamma_i}{\kappa_i - \kappa_{i-p-1}}\nu^p_{i-\frac{p-1}{2},i}(\varphi) & \text{if } 1 + \frac{p+1}{2} \le i \le n - \frac{p+1}{2}, \\ \nu^p_{n-p,i}(\varphi) & \text{if } n - \frac{p+1}{2} < i \le n. \end{cases} \tag{B.7}$$

Finally we note that these coefficient functionals agree with those defined in [39] in the case of uniform knots, except for the coefficients near the boundaries. This is due to the fact that the author did not use the Greville abscissas in that work.

Summing up, the application of the quasi-interpolation operator consists of three steps:

1. Compute the $p+1$ coefficients $\lambda_{j,k}(\varphi)$ of the local interpolating polynomials, see (B.3) and (B.4),
2. create the required spline coefficients of these polynomials via (B.5) (one or two for each inner polynomial with even and odd degree p, respectively, and $\lceil \frac{p}{2} \rceil$ for each of the boundary polynomials), and
3. evaluate the coefficient functionals according to (B.6) and (B.7).

These three steps can be combined into a single formula of the form (11).

Appendix C. Analysis of other techniques for matrix assembly

We briefly analyze the computational effort needed for assembling the mass matrix via Gauss quadrature (GQ), element-wise Gauss quadrature with sum factorization (EGS), interpolation and look-up (IL), weighted quadrature (WQ), and global Gauss quadrature with sum factorization (GGS).

Appendix C.1. Gauss quadrature (GQ)

After introducing a tensor-product grid of Gauss nodes $\hat{\boldsymbol{g}}_{\boldsymbol{k}} = (\hat{g}_{1,k_1}, \hat{g}_{2,k_2}, \hat{g}_{3,k_3})$ with associated quadrature weights $\gamma_{\boldsymbol{k}} = \gamma_{1,k_1}\gamma_{2,k_2}\gamma_{3,k_3}$, we arrive at the expression

$$M_{\boldsymbol{ij}} \approx \sum_{\boldsymbol{k}} [\hat{\beta}_{\boldsymbol{i}}(\hat{\boldsymbol{g}}_{\boldsymbol{k}})][\hat{\beta}_{\boldsymbol{j}}(\hat{\boldsymbol{g}}_{\boldsymbol{k}})][w(\hat{\boldsymbol{g}}_{\boldsymbol{k}})\gamma_{\boldsymbol{k}}]. \tag{C.1}$$

Note that we enumerate the Gauss nodes by using a global index $\boldsymbol{k} = (k_1, k_2, k_3)$.

The number of Gauss nodes per univariate element will be denoted by m. As described in Section 8.1, one has to use as many as $m = p + 1$ Gauss nodes, in order to make sure that the quadrature error does not spoil the optimal rate of convergence. Reduced Gauss quadrature, which uses fewer Gauss nodes, has also been studied in the literature.

The direct evaluation of (C.1) proceeds as follows:

1. We firstly precompute and store the terms enclosed by the square brackets for all the relevant $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$, which requires about $2n^3m^3(p+1)^3 + (n-p)^3m^3$ flops.
2. Next, we visit n^3 instances of the first index $\boldsymbol{i}$.
3. For each $\boldsymbol{i}$, we visit $C_2(i_1) \cdot C_2(i_2) \cdot C_2(i_3)$ instances of the second index $\boldsymbol{j}$.
4. For each pair $(\boldsymbol{i}, \boldsymbol{j})$, we visit all elements in the support intersection of the two tensor-product B-splines $\hat{\beta}_{\boldsymbol{i}}$ and $\hat{\beta}_{\boldsymbol{j}}$, taking $E_2(i_1, j_1) \cdot E_2(i_2, j_2) \cdot E_2(i_3, j_3)$ elements into account.
5. For each element, we add the product of the terms enclosed by square brackets at m^3 associated Gauss nodes to the approximate value of the integral. It needs 3 flops per node.

Summing up, as n increases, the number of flops per dof tends to the value

$$A_{\mathrm{GQ}} = 3m^3\bar{C}_2^3\bar{E}_2^3 + 2m^3(p+1)^3 + m^3 = 3(p+1)^9 + 2(p+1)^6 + (p+1)^3 = \mathcal{O}(p^9).$$

Appendix C.2. Element-wise Gauss quadrature with sum factorization (EGS)

The work described in [3] relies on mesh element-based assembly, which is a well-established technology in traditional finite element analysis. Each mesh element contributes a local mass matrix of size $(p+1)^3 \times (p+1)^3$ to the full mass matrix.

More precisely, when considering the summation over the mesh elements, which are indexed by $\boldsymbol{\ell} = (\ell_1, \ell_2, \ell_3)$, the sum in (C.1) is transformed into

$$M_{\boldsymbol{ij}} \approx \sum_{\boldsymbol{\ell}} \underbrace{\sum_{k_3} [\hat{\beta}_{3,i_3}(\hat{g}_{3,k_3})\hat{\beta}_{3,j_3}(\hat{g}_{3,k_3})\gamma_{3,k_3}] \underbrace{\sum_{k_2} [\hat{\beta}_{2,i_2}(\hat{g}_{2,k_2})\hat{\beta}_{2,j_2}(\hat{g}_{2,k_2})\gamma_{2,k_2}] \underbrace{\sum_{k_1} [\hat{\beta}_{1,i_1}(\hat{g}_{1,k_1})\hat{\beta}_{1,j_1}(\hat{g}_{1,k_1})\gamma_{1,k_1}] w(\hat{\boldsymbol{g}}_{\boldsymbol{k}})}_{= M^{\mathrm{EGS},1}_{\boldsymbol{\ell},i_1 j_1 k_2 k_3}}}_{= M^{\mathrm{EGS},2}_{\boldsymbol{\ell},i_1 i_2 j_1 j_2 k_3}}}_{= M^{\mathrm{EGS},3}_{\boldsymbol{\ell},i_1 i_2 i_3 j_1 j_2 j_3}},$$

where we introduce auxiliary tensors $\boldsymbol{M}^{\mathrm{EGS},1}_{\boldsymbol{\ell}}$, $\boldsymbol{M}^{\mathrm{EGS},2}_{\boldsymbol{\ell}}$ and $\boldsymbol{M}^{\mathrm{EGS},3}_{\boldsymbol{\ell}}$ for each mesh element. The terms enclosed by the square brackets are pre-computed (with negligible computational effort) to speed up the computation.

The first (innermost) loop, which computes the first auxiliary tensors for all mesh elements, evaluates $(n-p)^3 \cdot (p+1) \cdot (p+1) \cdot m \cdot m$ elements $M^{\mathrm{EGS},1}_{\boldsymbol{\ell},i_1 j_1 k_2 k_3}$ by summing over m indices k_1 and requires 2 flops per term. The evaluation of the second auxiliary tensors with $(n-p)^3 \cdot (p+1) \cdot (p+1) \cdot (p+1) \cdot (p+1) \cdot m$ elements $M^{\mathrm{EGS},2}_{\boldsymbol{\ell},i_1 i_2 j_1 j_2 k_3}$ proceeds by summing over m indices k_2 and needs 2 flops per term. Finally we evaluate the local mass matrices with $(n-p)^3 \cdot (p+1) \cdot (p+1) \cdot (p+1) \cdot (p+1) \cdot (p+1) \cdot (p+1)$ elements $M^{\mathrm{EGS},3}_{\boldsymbol{\ell},i_1 i_2 i_3 j_1 j_2 j_3}$ by summing over m indices k_3, requiring another 2 flops per term. In addition, accumulating all these local matrices to the global matrix costs $(n-p)^3 \cdot (p+1)^6$ flops. Summing up, the total number of flops evaluates to

$$T_{\mathrm{EGS}} = (n-p)^3\big(2m^3(p+1)^2 + 2m^2(p+1)^4 + 2m(p+1)^6 + (p+1)^6\big) = \mathcal{O}(n^3 p^7),$$

which brings the asymptotic number of flops per dof to

$$A_{\mathrm{EGS}} = (p+1)^5(2p^2 + 7p + 7) = \mathcal{O}(p^7).$$

Appendix C.3. Interpolation and Look-up (IL)

The interpolation and look-up method [32] consists of three steps: Building the look-up tables, spline projection and matrix assembly.

Building the look-up tables is not costly, which can be neglected compared to the overall computational cost. The spline projection step requires at most $6n^3(p+2)$ flops, which has been analyzed in Section 7. In the assembly step, the evaluation of the mass matrix (15) proceeds as follows:

1. We visit n^3 instances of the first index $\boldsymbol{i}$.
2. For each $\boldsymbol{i}$, we visit $C_2(i_1) \cdot C_2(i_2) \cdot C_2(i_3)$ instances of the second index $\boldsymbol{j}$.

3. For each pair $(\boldsymbol{i},\boldsymbol{j})$, we visit $C_3(i_1,j_1)\cdot C_3(i_2,j_2)\cdot C_3(i_3,j_3)$ instances of the third index $\boldsymbol{k}$.
4. For each pair $(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k})$, the products of the values of tri-product integrals multiplied with the associated weight needs 4 flops.

Summing up, the asymptotic number of flops per dof does not exceed

$$6(p+2)+4\bar{C}_2^3\bar{C}_3^3 = 2(54p^6+162p^5+216p^4+162p^3+72p^2+21p+8) = \mathcal{O}(p^6).$$

Appendix C.4. Weighted quadrature (WQ)

Weighted quadrature [15] introduces special rules for the numerical integration of integrals. These rules evaluate the integrand on a tensor grid of nodes

$$\hat{\boldsymbol{q}}_{\boldsymbol{k}} = (\hat{q}_{1,k_1},\hat{q}_{2,k_2},\hat{q}_{3,k_3})$$

and form a weighted sum with the associated tensor-product weights

$$\nu_{\boldsymbol{ik}} = \nu_{1,i_1k_1}\nu_{2,i_2k_2}\nu_{3,i_3k_3}.$$

When considering only basis functions that are sufficiently far away from the domain boundary, it has been shown that it suffices to use $2p+1$ quadrature points in the support of the univariate B-splines, which consist of the inner knots and midpoints of the $p+1$ knot spans. Nodes and weights are chosen such that the rule exactly evaluates tensor-product spline functions of degree p. These rules are precomputed and the computational effort is not significant as it scales with n.

Consequently, the matrix elements are approximated by expressions of the form

$$M_{\boldsymbol{ij}} \approx \sum_{\boldsymbol{k}} \hat{\beta}_{\boldsymbol{j}}(\hat{\boldsymbol{q}}_{\boldsymbol{k}})\, w(\hat{\boldsymbol{q}}_{\boldsymbol{k}})\, \nu_{\boldsymbol{ik}}. \tag{C.2}$$

Again one uses sum factorization for speeding up the evaluation. We rewrite the above equation as

$$M_{\boldsymbol{ij}} \approx \underbrace{\sum_{k_3}[\hat{\beta}_{3,j_3}(\hat{q}_{3,k_3})\nu_{3,i_3k_3}] \underbrace{\sum_{k_2}[\hat{\beta}_{2,j_2}(\hat{q}_{2,k_2})\nu_{2,i_2k_2}] \underbrace{\sum_{k_1}[\hat{\beta}_{1,j_1}(\hat{q}_{1,k_1})\nu_{1,i_1k_1}] w(\hat{\boldsymbol{q}}_{\boldsymbol{k}})}_{=M^{\mathrm{WQ},1}_{i_1j_1k_2k_3}}}_{=M^{\mathrm{WQ},2}_{i_1i_2j_1j_2k_3}}}_{=M^{\mathrm{WQ},3}_{i_1i_2i_3j_1j_2j_3}}, \tag{C.3}$$

and introduce three auxiliary tensors $\boldsymbol{M}^{\mathrm{WQ},1}$, $\boldsymbol{M}^{\mathrm{WQ},2}$ and $\boldsymbol{M}^{\mathrm{WQ},3}$ for evaluating the matrix elements, focusing on non-zero tensor elements and terms. The terms enclosed by the square brackets are pre-computed (with negligible computational effort) to speed up the computation.

The first (innermost) loop, which computes the first auxiliary tensor, evaluates

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot 2(n-p) \cdot 2(n-p)$$

non-zero elements $M^{\mathrm{WQ},1}_{i_1 j_1 k_2 k_3}$ by summing over $2E_2(i_1, j_1) - 1$ indices k_1 for each pair (i_1, j_1) and requires 2 flops per term. The evaluation of the second auxiliary tensor with

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot 2(n-p)$$

non-zero elements $M^{\mathrm{WQ},2}_{i_1 i_2 j_1 j_2 k_3}$ proceeds by summing over $2E_2(i_2, j_2) - 1$ indices k_2 for each pair (i_2, j_2) and needs 2 flops per term. Finally we evaluate the

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot \sum_{i_3=1}^{n} C_2(i_3)$$

non-zero elements $M^{\mathrm{WQ},3}_{i_1 i_2 i_3 j_1 j_2 j_3}$ by summing over $2E_2(i_3, j_3) - 1$ indices k_3 for each pair (i_3, j_3), requiring another 2 flops per term. Therefore, as n increases, the computational effort per dof is asymptotically equal to

$$A_{\mathrm{WQ}} = (2\bar{E}_2 - 1)(8\bar{C}_2 + 4\bar{C}_2^2 + 2\bar{C}_2^3) = 16p^4 + 48p^3 + 68p^2 + 44p + 14 = \mathcal{O}(p^4). \tag{C.4}$$

Appendix C.5. Global Gauss quadrature with sum factorization (GGS)

Recently it has been noted that Gauss quadrature can be further accelerated by avoiding the element-based assembly [12]. More precisely, one rewrites the mass matrix as

$$M_{\boldsymbol{ij}} \approx \underbrace{\sum_{k_3} [\hat{\beta}_{3,i_3}(\hat{g}_{3,k_3}) \hat{\beta}_{3,j_3}(\hat{g}_{3,k_3}) \gamma_{3,k_3}] \underbrace{\sum_{k_2} [\hat{\beta}_{2,i_2}(\hat{g}_{2,k_2}) \hat{\beta}_{2,j_2}(\hat{g}_{2,k_2}) \gamma_{2,k_2}] \underbrace{\sum_{k_1} [\hat{\beta}_{1,i_1}(\hat{g}_{1,k_1}) \hat{\beta}_{1,j_1}(\hat{g}_{1,k_1}) \gamma_{1,k_1}] w(\hat{\boldsymbol{g}}_{\boldsymbol{k}})}_{= M^{\mathrm{GGS},1}_{i_1 j_1 k_2 k_3}}}_{= M^{\mathrm{GGS},2}_{i_1 i_2 j_1 j_2 k_3}}}_{= M^{\mathrm{GGS},3}_{i_1 i_2 i_3 j_1 j_2 j_3}} \tag{C.5}$$

with the help of three auxiliary tensors $\boldsymbol{M}^{\mathrm{GGS},1}$, $\boldsymbol{M}^{\mathrm{GGS},2}$ and $\boldsymbol{M}^{\mathrm{GGS},3}$. Again, the terms enclosed by the square brackets are pre-computed (with negligible computational effort) to speed up the computation. Clearly, it suffices to consider only the non-zero tensor elements and terms in (C.5).

The first (innermost) loop, which computes the first auxiliary tensor, evaluates

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot (n-p)m \cdot (n-p)m$$

non-zero elements $M^{\mathrm{GGS},1}_{i_1 j_1 k_2 k_3}$ by summing over $mE_2(i_1, j_1)$ indices k_1 for each pair (i_1, j_1) and requires 2 flops per term. The evaluation of the second auxiliary tensor with

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot m(n-p)$$

non-zero elements $M^{\mathrm{GGS},2}_{i_1 i_2 j_1 j_2 k_3}$ proceeds by summing over $mE_2(i_2, j_2)$ indices k_2 for each pair (i_2, j_2) and needs 2 flops per term. Finally we evaluate the

$$\sum_{i_1=1}^{n} C_2(i_1) \cdot \sum_{i_2=1}^{n} C_2(i_2) \cdot \sum_{i_3=1}^{n} C_2(i_3)$$

non-zero elements $M^{\mathrm{GGS},3}_{i_1 i_2 i_3 j_1 j_2 j_3}$ by summing over $mE_2(i_3, j_3)$ indices k_3 for each pair (i_3, j_3), requiring another 2 flops per term. Summing up, the number of flops per dof tends to

$$2m^3 \bar{C}_2 \bar{E}_2 + 2m^2 \bar{C}_2^2 \bar{E}_2 + 2m \bar{C}_2^3 \bar{E}_2 = \mathcal{O}(p^5)$$

as n increases, which brings the asymptotic number of flops per dof to

$$A_{\mathrm{GGS}} = 2(p+1)^3(7p^2 + 9p + 3).$$

Appendix D. Analysis of other techniques for matrix-free applications

Appendix D.1. Weighted Quadrature

According to evaluation formula (C.2) of WQ, the $\boldsymbol{i}$-th item of the product of M and a vector $\boldsymbol{u}$ takes the form

$$v_{\boldsymbol{i}} = \sum_{\boldsymbol{j}} M_{\boldsymbol{ij}} u_{\boldsymbol{j}} \approx \sum_{\boldsymbol{j}} \sum_{\boldsymbol{k}} \hat{\beta}_{\boldsymbol{j}}(\hat{\boldsymbol{q}}_{\boldsymbol{k}}) w(\hat{\boldsymbol{q}}_{\boldsymbol{k}}) \nu_{\boldsymbol{ik}} u_{\boldsymbol{j}}, \tag{D.1}$$

which can be rearranged as

$$v_{\boldsymbol{i}} \approx \sum_{k_3} \nu_{3,i_3 k_3} \underbrace{\sum_{k_2} \nu_{2,i_2 k_2} \underbrace{\sum_{k_1} \nu_{1,i_1 k_1} w(\hat{q}_{1,k_1}, \hat{q}_{2,k_2}, \hat{q}_{3,k_3}) V^{(3)}_{k_1 k_2 k_3}}_{= V^{(4)}_{i_1 k_2 k_3}}}_{= V^{(5)}_{i_1 i_2 k_3}} \tag{D.2}$$

and

$$V^{(3)}_{k_1 k_2 k_3} = \sum_{j_3} \hat{\beta}_{3,j_3}(\hat{q}_{3,k_3}) \underbrace{\sum_{j_2} \hat{\beta}_{2,j_2}(\hat{q}_{2,k_2}) \underbrace{\sum_{j_1} \hat{\beta}_{1,j_1}(\hat{q}_{1,k_1}) u_{j_1 j_2 j_3}}_{= V^{(1)}_{j_2 j_3 k_1}}}_{= V^{(2)}_{j_3 k_1 k_2}}, \tag{D.3}$$

where five auxiliary tensors $\boldsymbol{V}^{(\ell)}$ $(\ell = 1, \ldots, 5)$ are introduced. It suffices to consider the non-zero tensor elements and terms. The first (innermost) loop evaluates $n \cdot n \cdot 2(n-p)$ non-zero elements $V^{(1)}_{j_2 j_3 k_1}$ by summing over p indices j_1 if $\hat{q}_{1,k_1}$ is an element boundary point or $p+1$ indices j_1 if $\hat{q}_{1,k_1}$ is an element midpoint, and requires 2 flops per term. The evaluation of the second auxiliary tensor with $n \cdot 2(n-p) \cdot 2(n-p)$ non-zero elements $V^{(2)}_{j_3 k_1 k_2}$ proceeds by summing over p or $p+1$ indices j_2 and needs 2 flops per term. The third loop evaluates $2(n-p) \cdot 2(n-p) \cdot 2(n-p)$ non-zero elements $V^{(3)}_{k_1 k_2 k_3}$ proceeds by summing over p or $p+1$ indices j_3 and requires 2 flops per term. The evaluation of the fourth auxiliary tensor with $n \cdot 2(n-p) \cdot 2(n-p)$ non-zero elements $V^{(4)}_{i_1 k_1 k_2}$ proceeds by summing over $2p+1$ indices k_1 and needs 3 flops per term. The fifth loop evaluates $n \cdot n \cdot 2(n-p)$ non-zero elements $V^{(5)}_{i_1 i_2 k_3}$ proceeds by summing over $2p+1$ indices k_2 and requires 2 flops per term. Finally we evaluate the $n \cdot n \cdot n$ non-zero elements $v_{\boldsymbol{i}}$ by summing over $2p+1$ indices k_3, requiring another 2 flops per term.

Summing up, the total number of flops amounts to

$$\mathcal{O}(n^3 p)$$

and the asymptotic number of flops per dof is

$$32(2p+1) = \mathcal{O}(p).$$

Appendix D.2. Global Gauss quadrature with sum factorization

The entries of matrix-vector products using GGS can be derived from (C.5), taking the from

$$v_{\boldsymbol{i}} = \sum_{\boldsymbol{j}} M_{\boldsymbol{ij}} u_{\boldsymbol{j}} \approx \sum_{k_3} [\hat{\beta}_{3,i_3}(\hat{g}_{3,k_3})\gamma_{3,k_3}] \underbrace{\sum_{k_2} [\hat{\beta}_{2,i_2}(\hat{g}_{2,k_2})\gamma_{2,k_2}] \underbrace{\sum_{k_1} [\hat{\beta}_{1,i_1}(\hat{g}_{1,k_1})\gamma_{1,k_1}] w(\hat{\boldsymbol{g}}_{\boldsymbol{k}}) V^{(3)}_{k_1 k_2 k_3}}_{= V^{(4)}_{i_1 k_2 k_3}}}_{= V^{(5)}_{i_1 i_2 k_3}} \tag{D.4}$$

and

$$V^{(3)}_{k_1 k_2 k_3} = \sum_{j_3} \hat{\beta}_{3,j_3}(\hat{g}_{3,k_3}) \underbrace{\sum_{j_2} \hat{\beta}_{2,j_2}(\hat{g}_{2,k_2}) \underbrace{\sum_{j_1} \hat{\beta}_{1,j_1}(\hat{g}_{1,k_1}) u_{j_1 j_2 j_3}}_{= V^{(1)}_{j_2 j_3 k_1}}}_{= V^{(2)}_{j_3 k_1 k_2}},$$

where five auxiliary tensors $\boldsymbol{V}^{(\ell)}$ $(\ell = 1, \ldots, 5)$ are introduced. Again, the terms enclosed by the square brackets are pre-computed (with negligible computational effort) to speed up the computation.

It suffices to consider the non-zero tensor elements and terms. The first (innermost) loop evaluates $n \cdot n \cdot m(n-p)$ non-zero elements $V^{(1)}_{j_2 j_3 k_1}$ by summing over $(p+1)$ indices

j_1 and requires 2 flops per term. The evaluation of the second auxiliary tensor with $n \cdot m(n-p) \cdot m(n-p)$ non-zero elements $V^{(2)}_{j_3 k_1 k_2}$ proceeds by summing over $p+1$ indices j_2 and needs 2 flops per term. The third loop evaluates $m(n-p) \cdot m(n-p) \cdot m(n-p)$ non-zero elements $V^{(3)}_{k_1 k_2 k_3}$ proceeds by summing over $p+1$ indices j_3 and requires 2 flops per term. The evaluation of the fourth auxiliary tensor with $n \cdot m(n-p) \cdot m(n-p)$ non-zero elements $V^{(4)}_{i_1 k_2 k_3}$ proceeds by summing over $m(p+1)$ indices k_1 and needs 3 flops per term. The fifth loop evaluates $n \cdot n \cdot m(n-p)$ non-zero elements proceeds by summing over $m(p+1)$ indices k_2 and requires 2 flops per term. Finally we evaluate the $n \cdot n \cdot n$ non-zero elements $v_{\boldsymbol{i}}$ by summing over $m(p+1)$ indices k_3, requiring another 2 flops per term.

Summing up, the total number of flops amounts to

$$\mathcal{O}(n^3 p^4)$$

and the number of flops per dof is asymptotically equal to

$$(p+1)^2(5p^2+14p+13) = \mathcal{O}(p^4).$$

References

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

[2] M. Ainsworth, G. Andriamaro, and O. Davydov. Bernstein–Bézier finite elements of arbitrary order and optimal assembly procedures. *SIAM Journal on Scientific Computing*, 33(6):3087–3109, 2011.

[3] P. Antolin, A. Buffa, F. Calabrò, M. Martinelli, and G. Sangalli. Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization. *Computer Methods in Applied Mechanics and Engineering*, 285:817–828, 2015.

[4] F. Auricchio, F. Calabrò, T.J.R. Hughes, A. Reali, and G. Sangalli. A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 249:15–27, 2012.

[5] F. Auricchio, L. Beirão da Veiga, T.J.R. Hughes, A. Reali, and G. Sangalli. Isogeometric collocation methods. *Mathematical Models and Methods in Applied Sciences*, 20(11):2075–2107, 2010.

[6] M. Bartoň, R. Ait-Haddou, and V.M. Calo. Gaussian quadrature rules for C^1 quintic splines with uniform knot vectors. *Journal of Computational and Applied Mathematics*, 322:57–70, 2017.

[7] M. Bartoň and V.M. Calo. Gaussian quadrature for splines via homotopy continuation: rules for C^2 cubic splines. *Journal of Computational and Applied Mathematics*, 296:709–723, 2016.

[8] M. Bartoň and 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 Engineering*, 305:217–240, 2016.

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

[10] Y. Bazilevs, L. Beirão da Veiga, J.A. Cottrell, T.J.R. Hughes, and G. Sangalli. Isogeometric analysis: approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences*, 16(07):1031–1090, 2006.

[11] C. De Boor. Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS)*, 5(2):173–182, 1979.

[12] A. Bressan and S. Takacs. Sum factorization techniques in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 352:437–460, 2019.
[13] F. Calabrò and C. Manni. The choice of quadrature in NURBS-based isogeometric analysis. In *Proc. of the 3rd South-East European Conference on Computational Mechanics (SEECCM)*, pages 260–267. The National Technical University of Athens, 2013.
[14] F. Calabrò, C. Manni, and F. Pitolli. Computation of quadrature rules for integration with respect to refinable functions on assigned nodes. *Applied Numerical Mathematics*, 90:168–189, 2015.
[15] F. Calabrò, G. Sangalli, and M. Tani. Fast formation of isogeometric Galerkin matrices by weighted quadrature. *Computer Methods in Applied Mechanics and Engineering*, 316:606–622, 2017.
[16] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.
[17] L. Beirão da Veiga, A. Buffa, J. Rivas, and G. Sangalli. Some estimates for h–p–k-refinement in isogeometric analysis. *Numerische Mathematik*, 118(2):271–305, 2011.
[18] L. Beirão da Veiga, A. Buffa, G. Sangalli, and R. Vázquez. Mathematical analysis of variational isogeometric methods. *Acta Numerica*, 23:157–287, 2014.
[19] F. Fahrendorf, L. De Lorenzis, and H. Gomez. Reduced integration at superconvergent points in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 328:390–410, 2018.
[20] G+Smo: Geometry + Simulation modules. `github.com/gismo/`.
[21] H. Gomez and L. De Lorenzis. The variational collocation method. *Computer Methods in Applied Mechanics and Engineering*, 309:152–181, 2016.
[22] R.R. Hiemstra, F. Calabrò, D. Schillinger, and T.J.R. Hughes. Optimal and reduced quadrature rules for tensor product and hierarchically refined splines in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:966–1004, 2017.
[23] R.R. Hiemstra, G. Sangalli, M. Tani, F. Calabrò, and T.J.R. Hughes. Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 355:234–260, 2019.
[24] C. Hofreither. A black-box low-rank approximation algorithm for fast matrix assembly in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 333:311–330, 2018.
[25] T.J.R. Hughes. Mathematics of isogeometric analysis and applications: a status report. In *Mathematical Foundations of Isogeometric Analysis*, volume 33 of *Oberwolfach Reports*. Europ. Math. Soc., 2019. to appear.
[26] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, 2005.
[27] T.J.R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):301–313, 2010.
[28] K.A. Johannessen. Optimal quadrature for univariate and tensor product splines. *Computer Methods in Applied Mechanics and Engineering*, 316:84–99, 2017.
[29] B. Jüttler, U. Langer, A. Mantzaflaris, S.E. Moore, and W. Zulehner. Geometry + Simulation modules: Implementing isogeometric analysis. *Proceedings in Applied Mathematics and Mechanics*, 14(1):961–962, 2014.
[30] A. Karatarakis, P. Karakitsios, and M. Papadrakakis. GPU accelerated computation of the isogeometric analysis stiffness matrix. *Computer Methods in Applied Mechanics and Engineering*, 269:334–355, 2014.
[31] T. Lyche and L.L. Schumaker. Local spline approximation methods. *Journal of Approximation Theory*, 15(4):294–325, 1975.
[32] A. Mantzaflaris and B. Jüttler. Integration by interpolation and look-up for Galerkin-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:373–400, 2015.
[33] A. Mantzaflaris, B. Jüttler, B.N. Khoromskij, and U. Langer. Low rank tensor methods in Galerkin-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:1062–1085, 2017.
[34] J.M. Melenk, K. Gerdes, and C. Schwab. Fully discrete hp-finite elements: Fast quadrature. *Computer*

Methods in Applied Mechanics and Engineering, 190(32-33):4339–4364, 2001.
[35] M. Montardini, G. Sangalli, and L. Tamellini. Optimal-order isogeometric collocation at Galerkin superconvergent points. *Computer Methods in Applied Mechanics and Engineering*, 316:741–757, 2017.
[36] S.A. Orszag. Spectral methods for problems in complex geometrics. In *Numerical Methods for Partial Differential Equations*, pages 273–305. Elsevier, 1979.
[37] L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.
[38] D. Rypl and B. Patzák. Study of computational efficiency of numerical quadrature schemes in the isogeometric analysis. *Engineering Mechanics*, 304, 2012.
[39] P. Sablonnière. Univariate spline quasi-interpolants and applications to numerical analysis. *Rendiconti del Seminario Matematico*, 63(3):211–222, 2005.
[40] G. Sangalli and M. Tani. Matrix-free weighted quadrature for a computationally efficient isogeometric k-method. *Computer Methods in Applied Mechanics and Engineering*, 338:117–133, 2018.
[41] D. Schillinger, J.A. Evans, A. Reali, M.A. Scott, and T.J.R. Hughes. Isogeometric collocation: Cost comparison with Galerkin methods and extension to adaptive hierarchical NURBS discretizations. *Computer Methods in Applied Mechanics and Engineering*, 267:170–232, 2013.
[42] D. Schillinger, S.J. Hossain, and T.J.R. Hughes. Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 277:1–45, 2014.
[43] F. Scholz, A. Mantzaflaris, and B. Jüttler. Partial tensor decomposition for decoupling isogeometric Galerkin discretizations. *Computer Methods in Applied Mechanics and Engineering*, 336:485–506, 2018.
[44] M. Tani. A preconditioning strategy for linear systems arising from nonsymmetric schemes in isogeometric analysis. *Computers & Mathematics with Applications*, 74(7):1690–1702, 2017.