Optimal quadrature for univariate and tensor product splines

Article *in* Computer Methods in Applied Mechanics and Engineering · May 2016

DOI: 10.1016/j.cma.2016.04.030

1 author:

Kjetil Andre Johannessen
Norwegian University of Science and Technology

Optimal quadrature for univariate and tensor product splines

Kjetil André Johannessen
Department of Mathematical Sciences
Norwegian University of Science and Technology, Trondheim, Norway.
Department of Applied Mathematics
E-mail: Kjetil.Johannessen@math.ntnu.no

Abstract

Numerical integration is a core subroutine in many engineering applications, including the finite element method (FEM). Isogeometric analysis is a FEM technology that uses smooth B-spline and NURBS basis functions. Traditionally, Gaussian quadrature was used for numerical integration, but this yields suboptimal performance. This is attributed to the fact that Gaussian quadrature rules do not take inter-element smoothness of the spline basis into account, resulting in over-integration. The equations for exact quadrature are well known, but prove notoriously hard to solve due to their nonlinear nature. These generalized gauss rules were first introduced in [13] in the context of isogeometric analysis, where newton iteration was utilized to study and tabulate several cases. Later, homotopy continuation [6] was used as an alternative strategy for finding these rules. In this paper we describe an algorithm to generalize on both of these techniques. It is optimal in the sense that it will integrate a space of dimension n, using no more than $\frac{n+1}{2}$ quadrature points. The algorithm works on arbitrary nonuniform knot vectors of any polynomial degree and continuity, and is demonstrated on polynomial orders up to 15. It extends to 2D and 3D integrals by tensor product.

1 Introduction

Isogeometric analysis was introduced in [12] to bridge the gap between design and analysis. It employs the use of smooth nonuniform rational B-splines (NURBS) as a basis for the finite element method. The smooth basis has a number of intrinsic advantages beyond modeling convenience. It has been shown to have superior spectral properties [12] and allows for the construction of compatible spline spaces forming de Rham diagrams [7, 11, 14]. The smooth basis does, however, have some drawbacks as well. In particular, there were no general quadrature schemes to integrate piecewise smooth polynomials available. The proposed solution was to use Gaussian quadrature rules which were designed to integrate C^{-1} piecewise polynomials. Since discontinuous piecewise polynomials form a superspace of the smooth piecewise polynomials, integration was still exact, but at suboptimal performance. A number of authors [15, 17, 18] have commented on the expensive quadrature in isogeometric analysis.

There has been much research performed into finding optimal quadrature points for these spaces, but solutions have only been published on a subset of spaces. For instance [13] creates rules for certain polynomial degrees and continuities up to 5 knot spans suggesting to partition the global integration domain into integration-ranges or macro elements. Rules for C^1 quintic splines on uniform knot vectors was presented in [5], while rules for C^1 cubic splines on stretched knot vectors was provided in [2], which was extended to C^2 cubic splines in [6]. The authors in [4] considers a local computational model giving *nearly* optimal quadrature points, but is limited to certain knot vectors.

In this paper we give an algorithm which computes the optimal quadrature rules for spline integrands. It is demonstrated to work for any nonuniform knot vector of arbitrary degree and continuity. The paper is outlined as follows. In section 2 we establish notation and present B-spline functions. In section 3 we introduce the nonlinear equations for exact integration and provide a good initial guess to solve these with Newton iteration. The algorithm is shown to work on several knot vectors, in particular it is shown to always converge for uniform knots of maximum continuity. In section 4, we extend the algorithm to work on any general knot vectors by homotopy continuation, or continuous deformation of the knot vector. We provide some numerical results for a handful of spline spaces in section 5, before summarizing and concluding in section 6 and 7.

2 B-spline functions

In this section we briefly introduce B-spline basis functions. It is mainly to establish notation, as a more comprehensive introduction can be found for instance in [9]. Consider a knot vector of nondecreasing knots

$$\boldsymbol{\tau} = [\tau_1, \tau_2, ..., \tau_{n+p+1}], \quad \tau_{i+1} \geq \tau_i. \tag{1}$$

We can establish a set of basis functions from this knot vector by the recursive formula

$$\begin{aligned} N_{i,p,\boldsymbol{\tau}}(\xi) &= \frac{\xi - \tau_i}{\tau_{i+p} - \tau_i} N_{i,p-1,\boldsymbol{\tau}}(\xi) + \frac{\tau_{i+p+1} - \xi}{\tau_{i+p+1} - \tau_{i+1}} N_{i+1,p-1,\boldsymbol{\tau}}(\xi) \\ N_{i,0,\boldsymbol{\tau}}(\xi) &= \begin{cases} 1 & \text{if } \xi \in [\tau_i, \tau_{i+1}) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

where we define fractions such that $\frac{0}{0} := 0$. The functions $\{N_{i,p,\boldsymbol{\tau}}\}_{i=1}^n$ satisfy all the properties of a basis and span the space

$$\mathbb{S}^p_{\boldsymbol{\tau}} = \text{span}\{N_{i,p,\boldsymbol{\tau}}\} = \left\{\varphi \in L^2 \mid \ \varphi|_{\xi=\tau_i} \in C^{k_i}, \varphi|_{\xi\in(\tau_i,\tau_{i+1})} \in \mathbb{P}^p\right\} \tag{2}$$

where $\mathbb{P}^p$ is the space of polynomials of degree p, $k_i = p - m_i$ and m_i is the multiplicity of knot i, i.e. the number of times the knot τ_i appears in $\boldsymbol{\tau}$. Stated in plain text: $\mathbb{S}^p_{\boldsymbol{\tau}}$ is the space of all piecewise polynomials with a given smoothness between the different intervals. When the knot vector has $n + p + 1$ elements, we have $\dim(\mathbb{S}^p_{\boldsymbol{\tau}}) = n$.

The knot vector $\boldsymbol{\tau}$ is said to be *open* if the first and last knot is repeated exactly $p+1$ times, resulting in the basis being C^{-1} at the end points τ_1 and τ_{n+p+1}. In this work we often consider spaces of uniform continuity, say k, that is all internal knots are repeated the same number $m_i = p - k$ times. In this case, we have

$$\boldsymbol{\tau} = [\underbrace{\tau_1, ..., \tau_1}_{p+1 \text{ times}}, \underbrace{\tau_2, .., \tau_2}_{p-k \text{ times}}, \underbrace{\tau_3, ..., \tau_3}_{p-k \text{ times}}, ..., \underbrace{\tau_m, ..., \tau_m}_{p+1 \text{ times}}]. \tag{3}$$

This particular space of uniform continuity is referred to as $\mathbb{S}^p_k$, where we do not distinguish notationally between open and nonopen knot vectors. Note that in general, we may have mixed continuities, but uniform continuity is of great practical interest. We omit the knot vector subscript from the basis functions where there is little chance of confusion, i.e. $N_{i,p,\boldsymbol{\tau}}(\xi) = N_{i,p}(\xi)$.

Each interval $[\tau_i, \tau_{i+1}]$ with $\tau_{i+1} > \tau_i$ is denoted as a knot span, or element. We will refer to n_{el} as the number of elements.

2.1 B-spline derivatives and integrals

Since the space $\mathbb{S}^p_{\boldsymbol{\tau}}$ is comprised of piecewise polynomials, it may be unsurprising that the derivatives and integrals of any function in this space, can also be represented as a piecewise polynomials; of one degree lower and one degree higher respectively. In fact, one can write the derivative of a basis function as a linear combination of basis functions of one degree lower, over the same knot vector. For the integral expression [10], we need to augment the knot vector by padding the end of it. These expressions are given by

$$N'_{i,p,\boldsymbol{\tau}}(\xi) = \frac{p}{\tau_{i+p} - \tau_i} N_{i,p-1,\boldsymbol{\tau}}(\xi) - \frac{p}{\tau_{i+p+1} - \tau_{i+1}} N_{i+1,p-1,\boldsymbol{\tau}}(\xi) \tag{4}$$

$$\int_{\xi_0}^{\xi} N_{i,p,\boldsymbol{\tau}}(t) \, \mathrm{d}t = \frac{\tau_{i+p+1} - \tau_i}{p+1} \sum_{j=i}^{\tilde{n}} N_{j,p+1,\tilde{\boldsymbol{\tau}}}(\xi) \tag{5}$$

where

$$
\begin{aligned}
\boldsymbol{\tau} &= [\tau_1, \tau_2, ..., \tau_{n+p+1}] \\
\tilde{\boldsymbol{\tau}} &= [\tau_1, \tau_2, ..., \underbrace{\tau_{n+p+1}, ..., \tau_{n+p+1}}_{p+2 \text{ times}}]
\end{aligned}
$$

and $\tilde{n}$ is the dimension of the space spanned by the knot vector $\tilde{\boldsymbol{\tau}}$.

In particular, we note that

$$
\begin{aligned}
\int_{\mathbb{R}} N_{i,p,\boldsymbol{\tau}}(t)\ \mathrm{d}t &= \int_{\tau_i}^{\tau_{i+p+1}} N_{i,p,\boldsymbol{\tau}}(t)\ \mathrm{d}t \\
&= \frac{\tau_{i+p+1}-\tau_i}{p+1} \sum_{j=i}^{\tilde{n}} N_{j,p+1,\tilde{\boldsymbol{\tau}}}(\tau_{i+p+1}) - N_{j,p+1,\tilde{\boldsymbol{\tau}}}(\tau_i) \\
&= \frac{\tau_{i+p+1}-\tau_i}{p+1} \sum_{j=i}^{\tilde{n}} N_{j,p+1,\tilde{\boldsymbol{\tau}}}(\tau_{i+p+1}) \\
&= \frac{\tau_{i+p+1}-\tau_i}{p+1}. \qquad (6)
\end{aligned}
$$

3 Exact integration using quadrature

3.1 The governing equations

Exact integration on a space $\mathbb{S}^p_{\boldsymbol{\tau}}$ is characterized by

$$\int_{\mathbb{R}} \varphi(\xi)\ \mathrm{d}\xi = \sum_i w_i \varphi(\xi_i), \quad \forall \varphi \in \mathbb{S}^p_{\boldsymbol{\tau}} \qquad (7)$$

for some set of points ξ_i and scaling weights w_i. Assume now that we have a space of even dimension, i.e. that we have $2n$ basis functions. We note that in this case (7) is equivalent to

$$\int_{\mathbb{R}} N_{j,p}(\xi)\ \mathrm{d}\xi = \sum_i w_i N_{j,p}(\xi_i), \quad j = \{1, 2, ..., 2n\} \qquad (8)$$

where $\mathbb{S}^p_{\boldsymbol{\tau}} = \mathrm{span}\{N_{j,p}(\xi)\}$ and $N_{j,p}(\xi)$ is the usual B-spline basis functions. The system (8) is a set of $2n$ nonlinear equations. The unknowns in this system are the quadrature weights w_i and points ξ_i. While the existence and uniqueness [16] of these points have only been proven for certain types of knot vectors $\boldsymbol{\tau}$, there is strong numerical evidence that this is true in general as all examples considered in this work converged. With n unknown weights, n unknown points and $2n$ equations, it creates a well-defined square system with a unique solution. Equation (8) is linear in the unknown weights w_i, but polynomial in the quadrature points ξ_i. We cannot solve this exactly due to the Abel-Ruffini theorem which states that no solution to a general quintic polynomial can be expressed as radicals. It is however possible to solve this numerically by Newton iteration.

We write our equation system (8) as

$$F_j(\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\xi} \end{bmatrix}) = \sum_i w_i N_{j,p}(\xi_i) - \int_{\mathbb{R}} N_{j,p}(\xi)\mathrm{d}\xi = 0\ , \quad j = \{1, 2, ..., 2n\} \qquad (9)$$

or

$$\boldsymbol{F}(\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\xi} \end{bmatrix}) = \begin{bmatrix} N_{1,p}(\xi_1) & N_{1,p}(\xi_2) & N_{1,p}(\xi_3) & \dots & N_{1,p}(\xi_n) \\ N_{2,p}(\xi_1) & N_{2,p}(\xi_2) & N_{2,p}(\xi_3) & \dots & N_{2,p}(\xi_n) \\ \vdots & & & \ddots & \vdots \\ N_{2n,p}(\xi_1) & N_{2n,p}(\xi_2) & N_{2n,p}(\xi_3) & \dots & N_{2n,p}(\xi_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} - \begin{bmatrix} \int N_{1,p}(\xi)\mathrm{d}\xi \\ \int N_{2,p}(\xi)\mathrm{d}\xi \\ \vdots \\ \int N_{2n,p}(\xi)\mathrm{d}\xi \end{bmatrix}. \qquad (10)$$

The last term, i.e. $\int N_{j,p}(\xi)\mathrm{d}\xi$ is computed using (6). To solve the equation $\boldsymbol{F} = \boldsymbol{0}$ by Newton iteration, we need its Jacobian. This can be computed as

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{w}} = \begin{bmatrix} N_{1,p}(\xi_1) & N_{1,p}(\xi_2) & N_{1,p}(\xi_3) & \dots & N_{1,p}(\xi_n) \\ N_{2,p}(\xi_1) & N_{2,p}(\xi_2) & N_{2,p}(\xi_3) & \dots & N_{2,p}(\xi_n) \\ \vdots & & \ddots & & \vdots \\ N_{2n,p}(\xi_1) & N_{2n,p}(\xi_2) & N_{2n,p}(\xi_3) & \dots & N_{2n,p}(\xi_n) \end{bmatrix} \tag{11}$$

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} w_1 N'_{1,p}(\xi_1) & w_1 N'_{1,p}(\xi_2) & w_1 N'_{1,p}(\xi_3) & \dots & w_1 N'_{1,p}(\xi_n) \\ w_2 N'_{2,p}(\xi_1) & w_2 N'_{2,p}(\xi_2) & w_2 N'_{2,p}(\xi_3) & \dots & w_2 N'_{2,p}(\xi_n) \\ \vdots & & \ddots & & \vdots \\ w_n N'_{2n,p}(\xi_1) & w_n N'_{2n,p}(\xi_2) & w_n N'_{2n,p}(\xi_3) & \dots & w_n N'_{2n,p}(\xi_n) \end{bmatrix} \tag{12}$$

$$\partial \boldsymbol{F} = \left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{w}}, \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\xi}}\right] \in \mathbb{R}^{2n\times 2n}. \tag{13}$$

Denoting the collective set of unknowns $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\xi} \end{bmatrix}$ we may formulate Newton iteration as

$$\partial \boldsymbol{F}(\boldsymbol{z}^k)\delta \boldsymbol{z}^k = -\boldsymbol{F}(\boldsymbol{z}^k) \tag{14}$$

$$\delta \boldsymbol{z}^k = (\boldsymbol{z}^{k+1} - \boldsymbol{z}^k). \tag{15}$$

At each iterate level, equation (14) is solved for the update delta $\delta \boldsymbol{z}^k$ which is then added to the previous solution z^k to produce z^{k+1}. This is continued until some desired residual tolerance has been reached.

The system is sensitive to the choice of initial guess and without a proper starting point $\boldsymbol{z}^0$ we cannot, in general, expect to achieve convergence. One of the primary contributions of this work is to provide a good initial guess. We propose that the initial guess should be given as

$$w_i^0 = \int N_{2i,p}(\xi) + N_{2i+1,p}(\xi)\, \mathrm{d}\xi$$

$$\xi_i^0 = (\tau_{2i}^* + \tau_{2i+1}^*)/2 \tag{16}$$

(17)

with

$$\tau_i^* = \frac{1}{p}\sum_{j=i+1}^{i+p} \tau_j.$$

We will now give some reasoning behind the choice of these initial guesses. The interpolation points should be points of some significance to the basis. It has been shown in the literature that the Greville abscissae τ_i^* have a number of interesting properties [18, 9], so these already stand out as good candidates. Since we have twice as many Greville abscissae as we have quadrature points, we simply take the average of two neighbouring. The weights was chosen because they mirror the behaviour of the left-hand-side of (8) and again noting that we have twice as many left-hand-side components as we have quadrature weights, we sum the integral of two consecutive basis functions. We don't take the average as we want the integration of all functions $\sum_j \int N_{j,p}$ to equal that of the total weight $\sum_i w_i$.

The algorithm is outlined in Algorithm 1. It is almost complete, but it does not account for possible divergence in the Newton iteration. Typically one would like to add additional logic to handle this, such as put a maximum number of iterations on the computation of $\delta \boldsymbol{z}^k$ or break if $\partial \boldsymbol{F}$ becomes singular.

Algorithm 1 Generate Optimal Quadrature Rules

Require: p // Polynomial degree
Require: $\boldsymbol{\tau} = [\tau_1, ..., \tau_{2n+p+1}]$ // Knot vector with $2n$ basis functions
for $j \leftarrow 1$ **to** $2n$ **do**
  $\tau_j^* \leftarrow \sum_{i=j+1}^{i+p} \tau_i / p$ // Greville abscissae
  $I_j \leftarrow (\tau_{j+p+1} - \tau_j) / (p+1)$ // Exact integral $\int N_{j,p}(\xi) \, \mathrm{d}\xi$
end for
for $i \leftarrow 1$ **to** n **do**
  $\xi_i \leftarrow (\tau_{2i}^* + \tau_{2i-1}^*)/2$ // Initial guess
  $w_i \leftarrow I_{2i} + I_{2i-1}$ // Initial guess
end for
$\delta \boldsymbol{z} \leftarrow \infty$
while $\|\delta \boldsymbol{z}\| > \mathrm{TOL}$ **do** // Newton iteration loop
  for $j \leftarrow 1$ **to** $2n$ **do**
    for $i \leftarrow 1$ **to** n **do**
      $J_{j,i} \leftarrow N_{j,p}(\xi_i)$
      $J_{j,n+i} \leftarrow w_i N'_{j,p}(\xi_i)$ // $J = \partial \boldsymbol{F}$, jacobian matrix of $\boldsymbol{F}$
    end for
    $F_j \leftarrow \sum_{i=1}^{n} w_i N_{j,p}(\xi_i) - I_j$
  end for
  Solve $\boldsymbol{J} \delta \boldsymbol{z} = -\boldsymbol{F}$ for $\delta \boldsymbol{z} \in \mathbb{R}^{2n}$
  for $i \leftarrow 1$ **to** n **do**
    $w_i \leftarrow w_i + \delta z_i$
    $\xi_i \leftarrow \xi_i + \delta z_{n+i}$
  end for
end while
Return $(\xi_i, w_i), \quad i = \{1, ..., n\}$

3.2 Spline spaces of odd dimension

The previous section was written under the assumption that the spline space had an even number of basis functions. In order to get the quadrature rules for odd spaces, we simply create a superspace of even dimension and compute the quadrature points here. If the quadrature rules is exact for some $\mathbb{S}^p_{\hat{\boldsymbol{\tau}}} \supset \mathbb{S}^p_{\boldsymbol{\tau}}$, then clearly it will be exact for the space $\mathbb{S}^p_{\boldsymbol{\tau}}$ as well. Letting $\boldsymbol{\tau} = [\tau_1, ..., \tau_{2n+p}]$, it is quite easy to construct such a superspace $\hat{\boldsymbol{\tau}}$, by simply augmenting the knot vector $\boldsymbol{\tau}$, inserting any arbitrary single knot $\hat{\tau} \in [\tau_1, \tau_{2n+p}]$. We can choose any knot we like (including existing knots resulting in reduced continuity), but for numerical stability it is suggested inserting it in the center of the largest knot span.

3.3 Numerical tolerances

The work contained herein is based on numerical methods, and tolerances have to be introduced. The source code used to generate the quadrature points is provided in the appendix. We use three break criteria for stopping the Newton iteration:

1. Converges at $\|\delta \boldsymbol{z}^k\| < 10^{-10}$
2. Too many newton iteration steps at $i > 15$
3. Singular matrix $\partial \boldsymbol{F}$ if $\min_i(\xi_i) < \tau_1$ or $\max_i(\xi_i) > \tau_{2n+1}$

In practice, it turns out that it was the final one which was the hardest to reliably and efficiently detect. Naïve approaches such as taking the determinant or the diagonal elements after an LU-factorization are numerical unstable. Computing the condition number or rank on the matrix is more reliable, but when computed on a full matrix, end up completely dominating the computational time. Sparse options such as eigenvalue and condition number estimation were found to produce false positives. In the end, we propose a very coarse singularity condition (which is sufficient, but not necessary), stating that if $\xi_i < \tau_1$ or $\xi_i > \tau_{2n+1}$ for some i, then no basis function have support on this evaluation point and one would get an all-zero row in the matrix $\partial \boldsymbol{F}$. Any other singular matrix are simply ignored, and will ultimately fail to produce a convergent result after 15 iterations.

These values, in addition to the source code should be enough to reproduce all results. Do note, however, that there is no guarantee of this since low-level factors such as machine architecture (floating point arithmetic), linear algebra libraries and library/matlab versions may play some part. While there might be small deviations in the values of some tabulated numbers, we find it doubtful that the broad picture or conclusions would change.

3.4 Uniform knot vectors

In the following we demonstrate how this approach works on different knot vectors. First, we choose the trivial uniform knot vector of repeated integers

$$\boldsymbol{\tau} = [\underbrace{0, ..., 0}_{p-k \text{ times}}, \underbrace{1, .., 1}_{p-k \text{ times}}, \underbrace{2, ..., 2}_{p-k \text{ times}}, ..., \underbrace{n_{el}, ..., n_{el}}_{p-k \text{ times}}]. \tag{18}$$

It is unsurprising that the number of elements has hardly any impact on the convergence properties on such a regular knot. The continuity of the space however, has a more direct influence as to whether the scheme is converging or not. We have tabulated if the technique converges and, if so, how many iterations it takes to converge for all spline spaces $\mathbb{S}^p_k$ and continuities up to degree 15 in Table 1. When testing different continuities, we keep the knot spans (i.e. elements) the same, resulting in a different number of degrees of freedom. The location of the quadrature points for uniform knot vector of size $n_{el} = 9$ is depicted in Figure 2.

	$\mathbb{S}_0^p$	$\mathbb{S}_1^p$	$\mathbb{S}_2^p$	$\mathbb{S}_3^p$	$\mathbb{S}_4^p$	$\mathbb{S}_5^p$	$\mathbb{S}_6^p$	$\mathbb{S}_7^p$	$\mathbb{S}_8^p$	$\mathbb{S}_9^p$	$\mathbb{S}_{10}^p$	$\mathbb{S}_{11}^p$	$\mathbb{S}_{12}^p$	$\mathbb{S}_{13}^p$	$\mathbb{S}_{14}^p$
$\mathbb{S}_k^1$	2														
$\mathbb{S}_k^2$	5	1													
$\mathbb{S}_k^3$	5	1	5												
$\mathbb{S}_k^4$	5	5	5	4											
$\mathbb{S}_k^5$	5	2	5	5	5										
$\mathbb{S}_k^6$	6	6	5	5	5	5									
$\mathbb{S}_k^7$	6	6	6	5	5	5	5								
$\mathbb{S}_k^8$	7	10	6	5	5	5	5	5							
$\mathbb{S}_k^9$	9	7	-	6	5	6	5	5	5						
$\mathbb{S}_k^{10}$	-	-	-	8	6	5	5	5	6	6					
$\mathbb{S}_k^{11}$	-	-	-	-	-	5	6	5	6	6	6				
$\mathbb{S}_k^{12}$	-	-	-	-	-	6	6	6	6	6	6	6			
$\mathbb{S}_k^{13}$	-	-	-	-	-	6	6	7	6	6	6	6	6		
$\mathbb{S}_k^{14}$	-	-	-	-	-	-	7	6	8	6	6	6	7	7	
$\mathbb{S}_k^{15}$	-	-	-	-	-	-	-	6	9	8	7	7	7	7	7

Table 1: Number of iterations for Algorithm 1 to converge on a uniform knot vector of 128 knot spans, see (18). Keeping the number of elements n_{el} fixed, we vary polynomial degree p along the rows and continuity k along the columns. A dash "-" indicates that the method failed to converge. The table is triangular since the maximum continuity of a spline space of degree p is C^{p-1}. Notice in particular that the diagonal $\mathbb{S}_{p-1}^p$ is converging, that is it works for all spaces of maximum continuity.

3.5 Open uniform knot vectors

Next we consider an open knot vector with the first and last knot repeated $p+1$ times and all internal knots repeated $p-k$ times where k is the continuity, i.e.

$$\boldsymbol{\tau} = [\, \underbrace{0, ..., 0}_{p+1 \text{ times}}\,, \underbrace{1, .., 1}_{p-k \text{ times}}\,, \underbrace{2, ..., 2}_{p-k \text{ times}}\,, ..., \underbrace{n_{el}, ..., n_{el}}_{p+1 \text{ times}}]. \tag{19}$$

Again, we note that the number of knot spans n_{el} is not important for the convergence properties, and the results for all spline spaces of all continuities up to $p = 15$ using $n_{el} = 128$ is tabulated in Table 2.

3.6 Geometric knot vectors

Geometric knot vectors, often denoted as stretched knot vectors, are defined by a constant ratio on consecutive knots. Where uniform knot vectors keep the difference $\tau_{i+1} - \tau_i$ between knots constant, geometric knot vectors will keep the ratio $\tau_{i+1}/\tau_i = \alpha$ constant. We let the knot vector be given as

$$\boldsymbol{\tau} = [\underbrace{\alpha^{n_{el}-1}, ..., \alpha^{n_{el}-1}}_{p+1 \text{ times}}, \underbrace{\alpha^{n_{el}-2}, ..., \alpha^{n_{el}-2}}_{p-k \text{ times}}, ..., \underbrace{\alpha, ..., \alpha}_{p-k \text{ times}}, ..., \underbrace{1, ..., 1}_{p+1 \text{ times}}\,] \tag{20}$$

for some $\alpha < 1$. These knot vectors are of great practical interest as they are often used to resolve boundary layers in computational methods. This example is slightly more sensitive to the choice of α and n_{el}, and for brevity we choose to only tabulate $\alpha = \frac{4}{5}$ and $n_{el} = 128$ in Table 3.

4 Quadrature points as a function of the knot vector

As was demonstrated, the initial guess (16) produced a convergent series for a large number of knot vectors. We now describe an algorithm to ensure convergence of the rest of the knot vectors.

	$\mathbb{S}^p_0$	$\mathbb{S}^p_1$	$\mathbb{S}^p_2$	$\mathbb{S}^p_3$	$\mathbb{S}^p_4$	$\mathbb{S}^p_5$	$\mathbb{S}^p_6$	$\mathbb{S}^p_7$	$\mathbb{S}^p_8$	$\mathbb{S}^p_9$	$\mathbb{S}^p_{10}$	$\mathbb{S}^p_{11}$	$\mathbb{S}^p_{12}$	$\mathbb{S}^p_{13}$	$\mathbb{S}^p_{14}$
$\mathbb{S}^1_k$	2														
$\mathbb{S}^2_k$	5	5													
$\mathbb{S}^3_k$	5	5	5												
$\mathbb{S}^4_k$	5	5	5	5											
$\mathbb{S}^5_k$	5	5	5	5	5										
$\mathbb{S}^6_k$	6	6	6	6	6	5									
$\mathbb{S}^7_k$	6	6	7	7	6	6	5								
$\mathbb{S}^8_k$	7	-	-	8	7	6	6	6							
$\mathbb{S}^9_k$	-	-	-	-	-	7	6	6	6						
$\mathbb{S}^{10}_k$	-	-	-	-	-	-	8	7	6	6					
$\mathbb{S}^{11}_k$	-	-	-	-	-	-	-	8	7	7	6				
$\mathbb{S}^{12}_k$	-	-	-	-	-	-	-	-	9	8	7	7			
$\mathbb{S}^{13}_k$	-	-	-	-	-	-	-	-	-	-	9	7	7		
$\mathbb{S}^{14}_k$	-	-	-	-	-	-	-	-	-	-	-	-	8	8	
$\mathbb{S}^{15}_k$	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9

Table 2: Number of iterations for Algorithm 1 to converge on a open uniform knot vector of 128 knot spans with C^{-1} basis at the start and end, see (19). We let the polynomial degree p vary along the rows and the continuity k along the columns. A dash "-" indicates that the method failed to converge.

	$\mathbb{S}^p_0$	$\mathbb{S}^p_1$	$\mathbb{S}^p_2$	$\mathbb{S}^p_3$	$\mathbb{S}^p_4$	$\mathbb{S}^p_5$	$\mathbb{S}^p_6$	$\mathbb{S}^p_7$	$\mathbb{S}^p_8$	$\mathbb{S}^p_9$	$\mathbb{S}^p_{10}$	$\mathbb{S}^p_{11}$
$\mathbb{S}^1_k$	2											
$\mathbb{S}^2_k$	5	5										
$\mathbb{S}^3_k$	5	5	5									
$\mathbb{S}^4_k$	5	5	5	5								
$\mathbb{S}^5_k$	5	5	5	5	5							
$\mathbb{S}^6_k$	6	6	5	5	5	5						
$\mathbb{S}^7_k$	6	6	7	6	6	5	6					
$\mathbb{S}^8_k$	6	-	-	-	6	6	6	-				
$\mathbb{S}^9_k$	-	-	-	-	-	7	6	6	-			
$\mathbb{S}^{10}_k$	-	-	-	-	-	-	-	6	6	-		
$\mathbb{S}^{11}_k$	-	-	-	-	-	-	-	-	7	-	-	
$\mathbb{S}^{12}_k$	-	-	-	-	-	-	-	-	-	-	-	-

Table 3: Number of iterations to converge on a geometric open knot vector of 128 knot spans with C^{-1} basis at the start and end, see (20). A dash "-" indicates that the method failed to converge. The relation between consecutive knots were chosen to be $\alpha = 4/5$.

Inspired by the recent work in [6, 8] we reformulate the problem via homotopy continuation as the following. Let

$$\boldsymbol{G}(p, \boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\tau}) = \boldsymbol{0} \tag{21}$$

be our nonlinear equations for exact integration on the knot vector $\boldsymbol{\tau}$ of degree p. That is, for any fixed $\boldsymbol{\tau}$ and p, we have $\boldsymbol{G}(p, \boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\tau}) = \boldsymbol{F}(\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\xi} \end{bmatrix})$, with $\boldsymbol{F}$ given in (9). The interesting thing in this formulation is that we can let $\boldsymbol{\tau}$ vary. One can then think of $\boldsymbol{G}$ as polynomial in $\boldsymbol{\tau}$ and $(\boldsymbol{w}, \boldsymbol{\xi})$ as the roots of this polynomial. The solution roots will then depend continuously on $\boldsymbol{\tau}$ and any small change in the problem parameter $\boldsymbol{\tau}$ will result in a small change in the solution roots $(\boldsymbol{w}, \boldsymbol{\xi})$.

If Algorithm 1 diverges for some knot vector $\boldsymbol{\tau}$, we may reformulate the problem and search for quadrature points $(\hat{\boldsymbol{w}}, \hat{\boldsymbol{\xi}})$ on an "easier" knot vector $\hat{\boldsymbol{\tau}}$ and use these as initial guess for the harder knot $\boldsymbol{\tau}$. This might be done in a continuous fashion, where

$$\begin{aligned} \hat{\boldsymbol{\tau}}(t) &= t\boldsymbol{\tau} + (1-t)\boldsymbol{\tau}^{\mathrm{U}} \\ \boldsymbol{\tau} &= [\tau_1, \tau_2, ..., \tau_{m-1}, \tau_m] \\ \boldsymbol{\tau}^U &= [\tau_1, \frac{(m-1)\tau_1 + 1\tau_m}{m}, \frac{(m-2)\tau_1 + 2\tau_m}{m}, ..., \frac{1\tau_1 + (m-1)\tau_m}{m}, \tau_m] \end{aligned} \tag{22}$$

and $\boldsymbol{\tau}^{\mathrm{U}}$ is the uniform knot vector (over the domain $[\tau_1, \tau_m]$) of maximum regularity which we have shown always converges; see the diagonal in Table 1. Let $\boldsymbol{\tau}^U$ have the same number of components as $\boldsymbol{\tau}$. The t parameter can be thought of as a pseudo time-parameter, or measure of problem difficulty. With $t = 0$ we have the easiest uniform knot vector and with $t = 1$ we have the hardest problem, and the one which we ultimately are interested in. We solve the problem on successively harder and harder parameters t, using at each iteration the result from the previous easier iteration as the initial guess on our new problem.

Note that if multiple knots appear in $\boldsymbol{\tau}$ this poses no extra problem, as the knot vector $\hat{\boldsymbol{\tau}}$ will continuously deform into $\boldsymbol{\tau}$ and eventually collapse separate knots into multiple knots. If $\boldsymbol{\tau}$ does have duplicate knots, then $\boldsymbol{\tau}$ and $\boldsymbol{\tau}^U$ will have different number of knot spans (elements), but the same number of knots.

In order to minimize the number of subproblems to solve, we create a recursive bisection of the problem domain $t \in [0, 1]$. This is outlined in pseudo-code as Algorithm 2 and is best illustrated by an example. Assume that one has available the weights and points $(\boldsymbol{w}^0, \boldsymbol{\xi}^0)$ as well as the uniform knot vector $\hat{\boldsymbol{\tau}}(0) = \boldsymbol{\tau}^U$. If these initial guesses are not good enough to solve for $\hat{\boldsymbol{\tau}}(1) = \boldsymbol{\tau}$, then we use these as initial conditions for the problem on $\hat{\boldsymbol{\tau}}(\frac{1}{2})$. If the method converges, we progress on higher t by computing $\hat{\boldsymbol{\tau}}(\frac{3}{4})$ and if the method diverges, we try to solve for $\hat{\boldsymbol{\tau}}(\frac{1}{4})$. In the case that it fails for $\hat{\boldsymbol{\tau}}(\frac{1}{2})$, but converges for $\hat{\boldsymbol{\tau}}(\frac{1}{4})$, we use the result from the latter as initial guess in the former. If this fails to convergence, we try and solve for the center point between what we know $t = \frac{1}{4}$ and what we search for $t = \frac{1}{2}$, i.e. on the knot vector $\hat{\boldsymbol{\tau}}(\frac{3}{8})$, see Figure 1.

The number of subproblems t to solve for is tabulated in Table 4–6 for a selection of knot vectors.

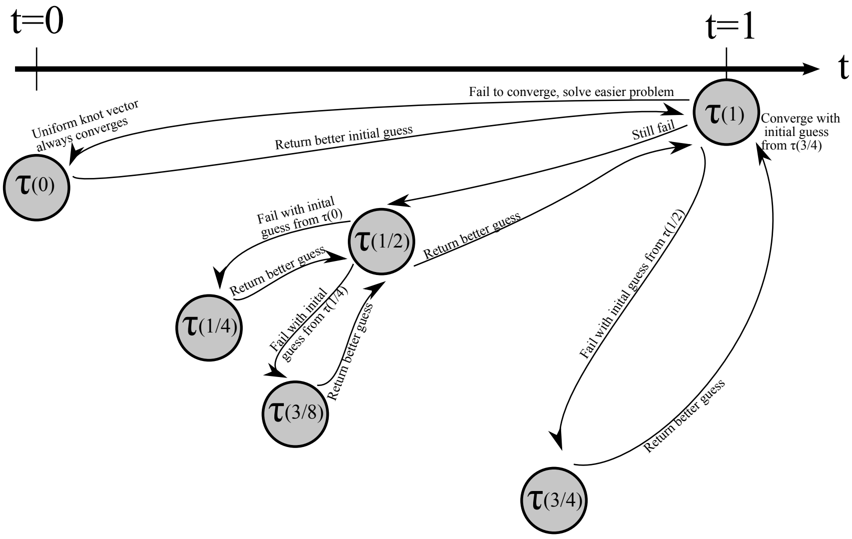


Figure 1: Example of the recursive calls of Algorithm 2

	$\mathbb{S}^p_0$	$\mathbb{S}^p_1$	$\mathbb{S}^p_2$	$\mathbb{S}^p_3$	$\mathbb{S}^p_4$	$\mathbb{S}^p_5$	$\mathbb{S}^p_6$	$\mathbb{S}^p_7$	$\mathbb{S}^p_8$	$\mathbb{S}^p_9$	$\mathbb{S}^p_{10}$	$\mathbb{S}^p_{11}$	$\mathbb{S}^p_{12}$	$\mathbb{S}^p_{13}$	$\mathbb{S}^p_{14}$	$\mathbb{S}^p_{15}$
$\mathbb{S}^8_k$	1	21	17	1	1	1	1	1								
$\mathbb{S}^9_k$	23	24	24	32	33	1	1	1	1							
$\mathbb{S}^{10}_k$	26	28	33	33	33	33	1	1	1	1						
$\mathbb{S}^{11}_k$	30	35	33	33	33	33	37	1	1	1	1					
$\mathbb{S}^{12}_k$	33	36	33	33	35	37	38	40	1	1	1	1				
$\mathbb{S}^{13}_k$	38	38	33	36	38	39	40	40	41	42	1	1	1			
$\mathbb{S}^{14}_k$	40	39	38	38	40	40	40	41	42	43	49	53	1	1		
$\mathbb{S}^{15}_k$	41	40	40	40	40	41	41	42	43	46	53	72	72	73	1	
$\mathbb{S}^{16}_k$	43	41	41	41	41	42	42	43	47	69	71	72	73	73	74	81

Table 4: Number of recursive calls to Algorithm 2 before returning the optimal quadrature points on a uniform open knot vector. This is the number of t-values we need to solve for before the algorithm converges. It is the same setup as in Table 2 using $n_{el} = 128$ elements and note that all convergent knot vectors in Table 2, are returning on the first recurisive call. The largest tabulated space is $\dim(\mathbb{S}^{16}_0) = 2049$.

	$\mathbb{S}^p_0$	$\mathbb{S}^p_1$	$\mathbb{S}^p_2$	$\mathbb{S}^p_3$	$\mathbb{S}^p_4$	$\mathbb{S}^p_5$	$\mathbb{S}^p_6$	$\mathbb{S}^p_7$	$\mathbb{S}^p_8$	$\mathbb{S}^p_9$	$\mathbb{S}^p_{10}$	$\mathbb{S}^p_{11}$	$\mathbb{S}^p_{12}$	$\mathbb{S}^p_{13}$
$\mathbb{S}^8_k$	1	1910	1631	1	1	1	1	1						
$\mathbb{S}^9_k$	2524	2190	1848	1617	1355	1	1	1	1					
$\mathbb{S}^{10}_k$	2871	2482	2120	1835	1623	1359	1	1	1	1				
$\mathbb{S}^{11}_k$	3348	2876	2415	2079	1829	1630	1369	1063	1	1	333			
$\mathbb{S}^{12}_k$	3586	3268	2762	2391	2081	1828	1630	1375	1070	1	1	339		
$\mathbb{S}^{13}_k$	3840	3509	3150	2729	2377	2089	1831	1635	1385	1079	855	587	351	
$\mathbb{S}^{14}_k$	4102	3739	3388	3120	2717	2369	2084	1835	1639	1390	1083	859	596	362

Table 5: Number of recursive calls to Algorithm 2 before returning the optimal quadrature points on a geometric open knot vector. This is using $\alpha = \frac{9}{10}$ and $n_{el} = 64$, see section 3.6.

Algorithm 2 Iterations on the knot vector

Require: $\boldsymbol{G}(p, \boldsymbol{w}^0, \boldsymbol{\xi}^0, \boldsymbol{\tau}^0) = \boldsymbol{0}$ // i.e. $(\boldsymbol{w}^0, \boldsymbol{\xi}^0)$ should be the solution on the knot vector $\boldsymbol{\tau}^0$

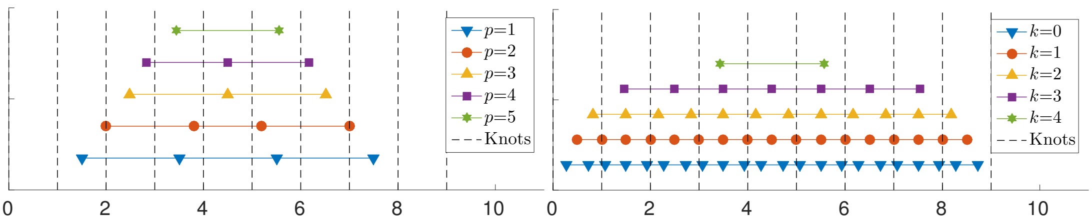
Function: $(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\tau})$ =`RecursiveKnotSearch`$(\boldsymbol{w}^0, \boldsymbol{\xi}^0, \boldsymbol{\tau}^0, \boldsymbol{\tau})$
$(\boldsymbol{w}^k, \boldsymbol{\xi}^k) \leftarrow (\boldsymbol{w}^0, \boldsymbol{\xi}^0)$
$\boldsymbol{\tau}^k \leftarrow \boldsymbol{\tau}^0$
while True **do**
  if Algorithm 1 converges on $\boldsymbol{\tau}$ with $(\boldsymbol{\xi}^k, \boldsymbol{w}^k)$ as initial guess **then**
    $(\boldsymbol{w}, \boldsymbol{\xi}) \leftarrow$ points and weights from Algorithm 1
    Return $(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\tau})$
  else
    $(\boldsymbol{w}^k, \boldsymbol{\xi}^k, \boldsymbol{\tau}^k) \leftarrow$`RecursiveKnotSearch`$(\boldsymbol{w}^k, \boldsymbol{\xi}^k, \boldsymbol{\tau}^k, \frac{\boldsymbol{\tau}^k + \boldsymbol{\tau}}{2})$
  end if
end while

	$\mathbb{S}^p_0$	$\mathbb{S}^p_1$	$\mathbb{S}^p_2$	$\mathbb{S}^p_3$	$\mathbb{S}^p_4$	$\mathbb{S}^p_5$	$\mathbb{S}^p_6$	$\mathbb{S}^p_7$	$\mathbb{S}^p_8$	$\mathbb{S}^p_9$	$\mathbb{S}^p_{10}$	$\mathbb{S}^p_{11}$
$\mathbb{S}^7_k$	1	1	1	1	1	1	1					
$\mathbb{S}^8_k$	1	10 232	8 674	1	1	1	1	1				
$\mathbb{S}^9_k$	13 482	11 928	10 164	8 660	7 293	1	1	1	1			
$\mathbb{S}^{10}_k$	14 873	13 379	11 821	10 156	8 677	7 307	1	1	1	1		
$\mathbb{S}^{11}_k$	16 358	14 797	13 288	11 788	10 162	8 703	7 326	5 990	1	1	1 659	
$\mathbb{S}^{12}_k$	17 845	16 216	14 662	13 253	11 789	10 171	8 712	7 343	6 011	1	1	1 676

Table 6: Number of recursive calls to Algorithm 2 before returning the optimal quadrature points on a geometric open knot vector. This is using $\alpha = \frac{9}{10}$ and $n_{el} = 128$, see section 3.6.

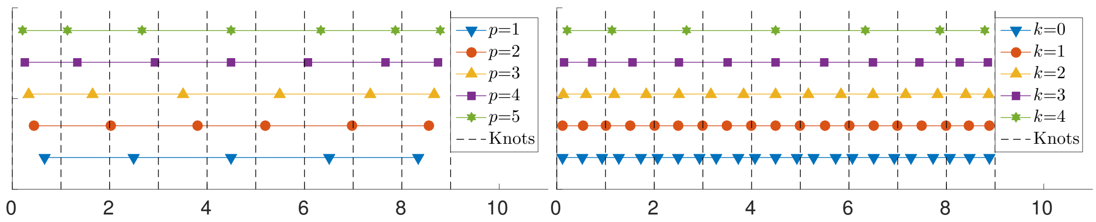
5 Numerical Results

In this section we display some of the quadrature point locations for different knot vectors. We show the location of the points for varying polynomial degrees and continuities on a uniform knot vector in Figure 2, an open knot vector in Figure 3 and a geometric knot vector in Figure 4.



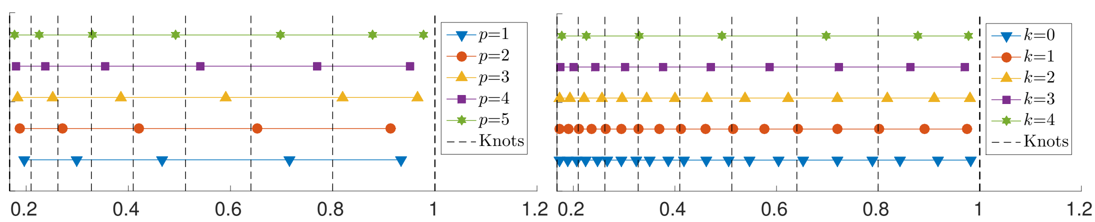
(a) $\mathbb{S}^p_{p-1}$ (b) $\mathbb{S}^5_k$

Figure 2: Location of quadrature points on uniform knot vector $\boldsymbol{\tau} = [0, 1, 2, ..., 9]$ (with multiplicity for the lower continuity case) when varying p for maximum continuity and varying k for $p = 5$. The quadrature points for the odd spaces $\mathbb{S}^4_3$ and $\mathbb{S}^2_1$ are not unique as they have been computed on a superspace given by $\hat{\boldsymbol{\tau}}$. We may choose any augmented knot vector we like as long as it contains all knots from the original one. In this example, we add the knot $\frac{9}{2}$ for symmetry.



(a) $\mathbb{S}^p_{p-1}$ (b) $\mathbb{S}^5_k$

Figure 3: Location of quadrature points on uniform open knot vector $\boldsymbol{\tau} = [0, 0, ..., 0, 1, 2, ..., 8, 9, 9, ..., 9]$ (with multiplicity for the lower continuity case) when varying p for maximum continuity and varying k for $p = 5$.



(a) $\mathbb{S}^p_{p-1}$ (b) $\mathbb{S}^5_k$

Figure 4: Location of quadrature points on a geometric open knot vector $\boldsymbol{\tau} = [\alpha^8, \alpha^8, ..., \alpha^8, \alpha^7, \alpha^6, ..., \alpha, 1, 1, ..., 1]$ with $\alpha = 0.9$ when varying p for maximum continuity and varying k for $p = 5$.

n_{el}	$\mathbb{S}^2_0$	$\mathbb{S}^2_1$	$\mathbb{S}^4_2$	$\mathbb{S}^4_3$	$\mathbb{S}^5_0$
5	1.320 ± 2.255	1.000 ± 0.000	5.880 ± 17.011	1.640 ± 4.504	32.740 ± 50.255
10	15.290 ± 29.933	1.000 ± 0.000	6.120 ± 24.617	1.000 ± 0.000	68.040 ± 100.217
20	53.790 ± 60.155	1.000 ± 0.000	23.590 ± 60.752	1.000 ± 0.000	155.450 ± 134.127
40	165.270 ± 111.065	1.600 ± 6.000	70.440 ± 149.699	1.000 ± 0.000	458.940 ± 279.209

n_{el}	$\mathbb{S}^5_4$	$\mathbb{S}^9_2$	$\mathbb{S}^9_8$	$\mathbb{S}^{12}_2$	$\mathbb{S}^{12}_{11}$
5	3.670 ± 13.383	55.970 ± 35.518	23.740 ± 24.774	81.120 ± 62.762	36.370 ± 51.511
10	2.760 ± 10.545	89.190 ± 46.633	26.610 ± 27.428	126.080 ± 77.523	44.210 ± 31.343
20	3.170 ± 18.825	202.160 ± 126.655	26.480 ± 34.507	273.190 ± 151.510	47.040 ± 27.919
40	6.710 ± 30.718	491.740 ± 217.380	16.290 ± 27.548	663.080 ± 272.988	43.670 ± 29.511

Table 7: Mean and standard deviation of recursive calls to Algorithm 2 before returning the optimal quadrature points on random knot vector. The values are reported as $a \pm \sigma$ with a being the mean value and σ the standard deviation computed from 100 simulations. Explicit rules for the construction of the random knot vector can be found in section 5.1. All knot vectors generated the optimal quadrature points and weights, and none diverged or produced an infinite recursion.

5.1 Random knot vector

To illustrate the generality of the technique, we apply it to a set of random knot vectors of mixed continuity. The knot vectors are chosen in the following manner

1. Choose global polynomial degree p, continuity k and number of elements n_{el}
2. Choose $n_{el} + 1$ knots from a Gaussian distribution of mean 0 and standard deviation 10
3. Duplicate all knots to get a global maximum continuity k
4. Duplicate start and end knots to get an open knot vector
5. Pick 10 percent of of the interior knots from point 2, rounded up if 10% is not an integer
    - Knots picked at random from a uniform distribution
    - Knots picked multiple times are ignored, no new knot is picked to replace it
    - Set the continuity at these knots to a random number (uniform distribution) between 0 and $k - 1$
6. Repeat this process 100 times and store the number of recursive calls to Algorithm 2
7. Report mean and standard deviation in Table 7.

Of the generated knot vectors, all managed to returned the optimal quadrature points, and none diverged by producing an infinite recursion.

6 Cost and benefits

The primary motivation for this work was applications within isogeometric finite element analysis. For this comparison, let us look at the computational cost and benefit one gets from using these optimal quadrature points in such a setting. The algorithm presented in the previous section is the solution of a nonlinear global system of equations, and the obvious question is whether this is a costly operation compared to the assembly process or the solution of the linear system of equations. One has to remember that this is only global in a single parametric direction. It is in higher dimensional problems such as 2D and especially 3D, that these techniques makes the most difference.

Assume for consistency that we consider a spline space of degree $2n$ in each direction separately. For 3D problems, this means we have $(2n)^3$ unknowns, which completely dominates the cost of the solution of a $2n$ system, even a nonlinear one. For dynamic problems, the quadrature points can be reused during the entire simulation, but will have to be re-computed after refinement operations.

We list two important integrals which frequently appear in finite element problems

$$\int_\Omega N_{i,p}(\xi)N_{j,p}(\xi)\ \mathrm{d}\xi \tag{23}$$

$$\int_\Omega N'_{i,p}(\xi)N'_{j,p}(\xi)\ \mathrm{d}\xi \tag{24}$$

corresponding to the mass matrix and stiffness matrix. If one assumes $N_{i,p} \in \mathbb{S}^p_k$, then we have

$$N_{i,p}(\xi)N_{j,p}(\xi) \quad \in \quad \mathbb{S}^{2p}_k \tag{25}$$

$$N'_{i,p}(\xi)N'_{j,p}(\xi) \quad \in \quad \mathbb{S}^{2p-2}_{k-1}. \tag{26}$$

We request exact integration of all terms, that is we need to integrate up functions of the largest polynomial degree and lowest continuity, i.e. $\mathbb{S}^{2p}_{k-1}$. Note that the number of elements n_{el} is the same for all these spaces, but the dimension of them will differ. For any spline space with n_{el} elements we have that

$$\dim(\mathbb{S}^p_k) = (p-k)n_{el} + (k+1) \tag{27}$$

and we need half as many quadrature points for exact integration. For maximal continuity $k = p-1$, we have that in order to integrate $\mathbb{S}^{2p}_{k-1} = \mathbb{S}^{2p}_{p-2}$ exactly we need

$$n_{\text{quadrature}} = \frac{\dim(\mathbb{S}^{2p}_{p-2})}{2} = \frac{(p+2)n_{el} + (p-1)}{2} = \mathcal{O}\left(\frac{p+2}{2}n_{el}\right) \tag{28}$$

which is asymptotically half the number of points as traditional gauss integration

$$n_{\text{gauss}} = (p+1)n_{el} \tag{29}$$

This number is squared for 2D problems and cubed for 3D. Using exact integration in 3D we can get the number of integration points down to $\frac{1}{8}$ to that of gauss integration. By careful consideration of the different parts of each individual variational problem, one can slightly improve this number. For instance, if one integrates the mass matrix (23) and stiffness matrix (24) separately, then one can construct individual quadrature rules which are optimal for each part, instead of a global set of quadrature rules which is guaranteed exact on both. Finally, the total computation time at the gauss points can be further reduced by exploiting the tensor-product structure and performing sum factorization [3].

In the future we would like to investigate the possibilities of deliberate under-integration. The integrands arising in finite element methods (23)-(24) keep the lowest continuity of its two factors, while doubling its polynomial degree. This is attributing a lot to the high dimension of spline space of the integrand. While there exist research result on under-integration using inadequate polynomial degree, it is to the best of the authors knowledge, no similar results on using inadequate continuity. It is unknown what would happen if one were to apply the exact quadrature rules of $\mathbb{S}^{2p}_{2p-1}$ to functions in $\mathbb{S}^{2p}_{p-2}$. If it was proven that this could work without loss of convergence properties, it would mean a computational speedup of $(p+2)^3$ on assembly of 3D problems.

The use of selective- and under-integration in isogeometric analysis is already being used to combat numerical locking in nearly incompressible elasticity problems [1].

7 Conclusions and future work

We have constructed a computational method which generate the minimal number of quadrature points and weights on any given discretization spline space. It will always consist of no more than $\frac{n+1}{2}$ evaluation points, where n is the dimension of the space. In particular for 3D finite element methods based on smooth spline functions, the assembly process can be computed up to 10 times faster due to fewer evaluation points, while still confining to exact integration.

8 Acknowledgment

The author gratefully acknowledge the the financial support from the Norwegian Research Council and the industrial partners of the FSI-WT project (RCN grant no: 216465/E20): Meteorologisk Institutt, FFI, Statoil, Trønder Energi, Kjeller Vindteknikk and WindSim AS. We would also like to thank the reviewers for detailed response and many constructive comments.

A A matlab/octave implementation

Here we provide a complete matlab function to compute the optimal quadrature points. It takes as input a polynomial degree and any arbitrary knot vector and returns the optimal quadrature weights and points. The initial conditions `x0`, `w0` and `knot0` are optional. This is a combination of Algorithm 1 and 2 provided in this manuscript. It assumes the existence of a `BSpline(knot, p, t)` function, also provided here. This function returns a matrix of all B-spline basis function evaluated at all points, i.e. with input vector t_j it returns a sparse matrix `N(i,j)=`$N_i(t_j)$ as well as its derivatives `dN(i,j)=`$N_i'(t_j)$, also in sparse format.

In the web version of this article both functions are available for download, in addition to an optimized version of `BSpline(knot,p,t)` written in `C++` using the `mex` library.

Tables 1–3 where tabulated by the `it` value, while Tables 4–7 were tabulated using the `rec` value, both returned from `GetOptimalQuadPoints`.

```
function [w, x, rec, it] = GetOptimalQuadPoints(knot, p, w0, x0, knot0, rec)
% intended use: [w, x] = GetOptimalQuadPoints(knot,p)

  n  = numel(knot)-p-1; % dimension of our spline space

  if mod(n,2)==1 % need to have a space of even dimension
    i = find(diff(knot) == max(diff(knot)));
    i = i(ceil(end/2));                        % insert new knot in middle of
    knot = sort([knot,mean(knot(i:i+1))]); % the largest, centermost knot span
    n = n+1;
  end

  % compute all greville points and integrals (used for initial guess)
  greville       = zeros(n,1);
  exact_integral = zeros(n,1);
  for i=1:n
    greville(i)       = sum(knot(i+1:i+p)) / p;
    exact_integral(i) = (knot(i+p+1)-knot(i))/(p+1);
  end

  if exist('x0') % if initial guess is provided, use these
    x = x0;
    w = w0;
  else             % else compute them based on greville points and integrals
    w   = (exact_integral(1:2:end) + exact_integral(2:2:end))  ;
    x   = (      greville(1:2:end) +       greville(2:2:end))/2;
    rec = 1;     % counter variable to count the number of recursive calls
  end

  newton_tol       = 1e-10;     % convergence tolerance
  newton_max_it    = 15;        % max iterations before divergence
  while true                    % recursive loop from algorithm 2
    for it = 1:newton_max_it    % newton iteration loop
      [N dN] = BSpline(knot, p, x);
      F      = N*w - exact_integral;
      dF     = [N, dN*diag(sparse(w))];

      dx = dF \ -F;
      w = w + dx(1:end/2);
      x = x + dx(end/2+1:end);

      % test for diverging (coarse heuristic, see section 3.3)
      if( min(x)<knot(1)   )     break;   end;
      if( max(x)>knot(end))      break;   end;

      % test for converging
      if(norm(dx)<newton_tol)    return;  end;
    end

    % at this point, newton iteration has diverged. solve recursively on easier knot
    if exist('knot0')
      [w, x, rec] = getOptimalQuadPoints((knot0 + knot)/2, p, w0, x0, knot0, rec);
      knot0       = (knot0 + knot)/2;
    else
      uniformKnot = linspace(knot(1),knot(end), n+p+1);
      [w, x, rec] = getOptimalQuadPoints(uniformKnot, p);
      knot0       = uniformKnot;
    end
    rec = rec + 1;
    x0 = x;
    w0 = w;
  end % loop up and start newton iteration with better initial guess
```

```
function [N dN] = BSpline(knot, p, t)
% function [N dN] = BSpline(knot, p, t)
%     parameters:
%         knot - the knot vector
%         p    - the polynomial order of the basis
%         t    - m component vector of points which is to be evaluated
%     returns:
%         N    - n by m matrix of the solution of all basis functions i
%                evaluated at all points xi(j), given in N(i,j)
%         dN   - n by m matrix of the solution of all derivative of the
%                basis functions i evaluated at all points xi(j), given
%                in dN(i,j)

% pad knot vector, so we always compute with C^{-1} at the start/end
knot = [knot(1)*ones(1,p), knot, knot(end)*ones(1,p)];
n   = numel(knot)-p-1;           % number of basis functions +2p
Ni  = ones( numel(t)*(p+1),1);
Nj  = ones( numel(t)*(p+1),1);
Nv  = zeros(numel(t)*(p+1),1);
dNv = zeros(numel(t)*(p+1),1);
for i=1:numel(t),
  if t(i)==knot(end-p)      % evaluate right end-point from the left
    mu = find(knot>=t(i), 1);
  else                      % else evlauate in the limit from the right
    mu = find(knot>t(i), 1);
  end
  if numel(mu)==0 || mu==1 % evaluation outside domain
    continue;
  end
  mu = mu-1;                % index of last non-zero basis function

  N  = 1;
  for q=1:p,
    k = mu-q+1:mu;
    R = zeros(q+1,q);
    R(1:q+2:end) = (knot(k+q) - t(i)    ) ./ (knot(k+q)-knot(k));
    R(2:q+2:end) = (t(i)      - knot(k)) ./ (knot(k+q)-knot(k));
    if p==q
      dR = zeros(q+1,q);
      dR(1:q+2:end) = -p ./ (knot(k+q)-knot(k));
      dR(2:q+2:end) =  p ./ (knot(k+q)-knot(k));
      dN = dR*N;
    end
    N = R*N;
  end
  Ni( (i-1)*(p+1)+1:i*(p+1)) = mu-p:mu;
  Nj( (i-1)*(p+1)+1:i*(p+1)) = i*ones(1, p+1);
  Nv( (i-1)*(p+1)+1:i*(p+1)) = N;
  dNv((i-1)*(p+1)+1:i*(p+1)) = dN;
end
N  = sparse(Ni,Nj, Nv, n, numel(t));
dN = sparse(Ni,Nj,dNv, n, numel(t));

N  =  N(p+1:end-p,:); % remove extra functions from padding the knot vector
dN = dN(p+1:end-p,:);
```

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. Isogeometric Analysis Special Issue.

[2] R. Ait-Haddou, M. Bartoň, and V.M. Calo. Explicit Gaussian quadrature rules for cubic splines with symmetrically stretched knot sequences. *Journal of Computational and Applied Mathematics*, 290:543–552, 2015.

[3] P. Antolin, A. Buffa, F. Calabró, 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. Calabró, 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*, 249–252:15–27, 2012. Higher Order Finite Element and Isogeometric Methods.

[5] M. Bartoň, R. Ait-Haddou, and V.M. Calo. Gaussian quadrature rules for C^1 quintic splines. *ArXiv e-prints*, March 2015.

[6] M. Bartoň and V.M. Calo. Gaussian quadrature for splines via homotopy continuation: rules for C^2 cubic splines. *ArXiv e-prints*, May 2015.

[7] A. Buffa, G. Sangalli, and R. Vázquez. Isogeometric methods for computational electromagnetics: B-spline and T-spline discretizations. *Journal of Computational Physics*, 257, Part B(0):1291–1320, 2014. Physics-compatible numerical methods.

[8] H. Cheng, V. Rokhlin, and N. Yarvin. Nonlinear optimization, quadrature, and interpolation. *SIAM J. on Optimization*, 9(4):901–923, April 1999.

[9] C. de Boor. On calculating with B-splines. *Journal of Approximation Theory*, 6(1):50–62, 1972.

[10] C. de Boor, T. Lyche, and L. L. Schumaker. On Calculating with B-Splines II. Integration. In L. Collatz, H. Werner, and G. Meinardus, editors, *Numerische Methoden der Approximationstheorie/Numerical Methods of Approximation Theory*, volume 30 of *International Series of Numerical Mathematics*, pages 123–146. Birkhäuser Basel, 1976.

[11] J.A. Evans and T.J.R. Hughes. Isogeometric divergence-conforming B-splines for the steady Navier-Stokes equations. *Math. Models Methods Appl. Sci.*, 23(8):1421–1478, 2013.

[12] 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, October 2005.

[13] 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. Computational Geometry and Analysis.

[14] K.A. Johannessen, M. Kumar, and T. Kvamsdal. Divergence-conforming discretization for Stokes problem on locally refined meshes using LR B-splines. *Computer Methods in Applied Mechanics and Engineering*, 293:38–70, 2015.

[15] A. G. Kravchenko, P. Moin, and K. Shariff. B-Spline Method and Zonal Grids for Simulations of Complex Turbulent Flows. *Journal of Computational Physics*, 151(2):757–789, 1999.

[16] C.A. Micchelli and A. Pinkus. Moment Theory for Weak Chebyshev Systems with Applications to Monosplines, Quadrature Formulae and Best One-Sided L^1-Approximation by Spline Functions with Fixed Knots. *SIAM Journal on Mathematical Analysis*, 8(2):206–230, 1977.

[17] K. Nordanger, R. Holdahl, A.M. Kvarving, A. Rasheed, and T. Kvamsdal. Implementation and comparison of three isogeometric Navier-Stokes solvers applied to simulation of flow past a fixed 2D NACA0012 airfoil at high Reynolds number. *Computer Methods in Applied Mechanics and Engineering*, 284:664–688, 2015. Isogeometric Analysis Special Issue.

[18] 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.