BERNSTEIN-BÉZIER FINITE ELEMENTS OF ARBITRARY ORDER AND OPTIMAL ASSEMBLY PROCEDURES

MARK AINSWORTH, GAELLE ANDRIAMARO, AND OLEG DAVYDOV

Abstract. Algorithms are presented that enable the element matrices for the standard finite element space, consisting of continuous piecewise polynomials of degree n on simplicial elements in $\mathbb{R}^d$, to be computed in optimal complexity $\mathcal{O}(n^{2d})$. The algorithms (i) take account of numerical quadrature; (ii) are applicable to non-linear problems; and, (iii) do not rely on pre-computed arrays containing values of one-dimensional basis functions at quadrature points (although these can be used if desired). The elements are based on Bernstein polynomials and are the first to achieve optimal complexity for the standard finite element spaces on simplicial elements.

1. Introduction

The classical finite element method, based on low order piecewise polynomial approximation using Lagrange shape functions in conjunction with mesh refinement, offers geometric flexibility but suffers from relatively poor resolution. The spectral method, on the other hand, achieves high resolution by employing a fixed mesh (often a single element) in conjunction with very high degree polynomial approximation, but suffers from a lack of geometric flexibility compared with the finite element method. The spectral/*hp*-version finite element method aims to combine the advantages of each approach by using high degree piecewise polynomials in a finite element setting [15, 22, 24]. Whilst the theory of the spectral/*hp*-version finite element method is quite well-established, the algorithmic details and practical implementation are not yet at the stage whereby the method can achieve its full potential [25].

One of the major bottlenecks of the method lies in the element level computations for the evaluation of the local stiffness matrices. The element stiffness matrices for degree n approximation in d dimensions contain $\binom{n+d}{n}^2$ entries which means that (even if each entry could be computed in $\mathcal{O}(1)$ operations) the overall cost of computing the element matrix is at least $\mathcal{O}(n^{2d})$. Of course, the cost of computing an individual entry is not just $\mathcal{O}(1)$ and numerical quadrature is often needed to approximate the entries. Any quadrature rule must use at least $\mathcal{O}(n^d)$ quadrature points, meaning that a naive implementation would result in a cost of $\mathcal{O}(n^d)$ per entry, and a prohibitively high overall cost of $\mathcal{O}(n^{3d})$ for the assembly of the element matrix.

The use of high order polynomials is typical of the spectral element method [5], where one routinely sees computations using polynomials of degree n in the order of 100s. The sum factorisation method goes back to Orszag [19] and is generally considered to be the main reason why high order spectral methods can be efficiently implemented [10, 15, 25].

1991 *Mathematics Subject Classification.* 65N30.

Key words and phrases. spectral/*hp* finite element; Bernstein polynomials.

Partial support of MA by the Engineering and Physical Sciences Research Council of Great Britain under grant EP/G036136/1, Numerical Algorithms and Intelligent Software for the Evolving HPC Platform, is gratefully acknowledged. The work of GA was supported in part by SORSAS.

The key property needed to utilise the sum factorisation method is a tensorial construction for the basis functions. For quadrilateral and hexahedral finite elements, the basis functions naturally have a tensor product structure and efficient procedures have been developed to take advantage of this fact [18]. However, simplicial elements do not have a tensor product structure and the natural bases for the corresponding finite element spaces are non-tensorial. Typically, for simplicial elements Lagrange bases are used in the case of very low order approximation, whilst hierarchic bases [1, 2, 6, 24] are used for higher orders of approximation. Although both choices are natural, neither has the key tensor product structure needed to exploit the sum factorisation technique.

Tensorial constructions for basis functions on triangular and tetrahedral elements are available [3, 8, 15, 26]. Of course, the effect of the non-tensorial nature of the underlying domain persists and manifests itself in a lack of symmetry in the basis functions (through the need to identify a preferred vertex in the case of triangular elements) and in the corresponding degrees of freedom. By employing the sum factorisation method in conjunction with such a tensorial element level basis, Karniadakis and Sherwin [15] obtain an algorithm for the assembly of the element matrix which achieves near optimal complexity $\mathcal{O}(n^{2d+1})$.

The desire to achieve optimal complexity prompted Eibner and Melenk [10] to make use of a larger local polynomial space than the standard space $\mathbb{P}_d^n$. The rather novel idea behind this approach is to use the additional variables to control the locations of the zeros of the basis functions. By arranging for the zeros to be located at nodes of the quadrature rule and again taking advantage of sum factorisation and a tensorial basis, the overall complexity is improved to the optimal count $\mathcal{O}(n^{2d})$. Unfortunately, this comes at the price of using a non-standard approximation space with larger numbers of degrees of freedom than the standard space (without a corresponding improvement in the rate of convergence).

The foregoing developments represent important advances in tackling the problem of realising efficient and practical high order finite elements on simplicial elements in optimal complexity. Nevertheless, the problem essentially remains unsolved and the following questions remain:

- can the optimal complexity of $\mathcal{O}(n^{2d})$ operations be achieved for the standard space $\mathbb{P}_d^n$?

and, if so,

- is this possible using a basis corresponding to a natural (e.g. symmetric) choice of degrees of freedom?

The purpose of the present work is to address these questions. The need to develop tensorial bases (in order to utilise sum factorisation) seems at odds with the non-tensorial nature of the underlying element geometry, and it is perhaps therefore rather surprising that *the answer to both questions is positive*.

The key idea is the use of Bernstein polynomials to construct a basis for the standard H^1-conforming finite element space consisting of continuous piecewise polynomials as described in Section 2. Although Bernstein-Bézier representations have a number of properties which make their use commonplace in computationally demanding applications such as CAGD and visualisation [11, 13, 14, 17], their use in finite element approximation has attracted virtually no attention [21] amongst the finite element community. Recently, the use of Bernstein-Bézier bases for high order finite element computation has been investigated by Kirby [16] in the case of piecewise constant data.

At first glance, the Bernstein-Bézier basis is reminiscent of the Lagrange basis on equally spaced nodes over the simplex. However, there is an essential difference in that the degrees of freedom for the Bernstein-Bézier finite element are *not* values

at these nodes (which is the case for Lagrange bases). Nonetheless, at least for conceptual purposes, one can regard the nodes as degrees of freedom. Viewed from this perspective, the Bernstein-Bézier elements are completely symmetrical and just as natural as the Lagrange elements typically found in finite element textbooks.

Of course, the critical factor is the complexity required to assemble the element matrices for the Bernstein-Bézier finite element and it is here that a remarkable property of the Bernstein polynomials comes into play. Although not based on a tensorial construction, the Bernstein-Bézier basis is rather unique in the respect that it possesses the key properties needed for the sum factorisation algorithm to be brought into play. This is exploited in Section 3 for the development of highly efficient algorithms for the computation of the *Bernstein-Bézier moments* of the coefficients in the underlying partial differential equation. The fact that the Bernstein polynomials are amenable to the sum factorisation approach is vital, but by itself is not sufficient to develop optimal element level algorithms. The remaining ingredient is another property whereby the product of two Bernstein polynomials is a (scaled) Bernstein polynomial. This property is exploited in Section 4 to develop algorithms for the assembly of the element matrices in $\mathcal{O}(n^{2d})$ operations.

The algorithms developed in Sections 3 and 4 are the first that enable the element matrices for the standard finite element space (continuous piecewise polynomials of degree n on simplicial elements in $\mathbb{R}^d$) to be assembled in optimal complexity $\mathcal{O}(n^{2d})$. In addition, our algorithms: (i) take account of numerical quadrature; (ii) are applicable to non-linear problems; and, (iii) do not rely on pre-computed arrays containing values of one-dimensional basis functions at quadrature points (although these can be used if desired). CPU timings are presented for each of the algorithms showing the predicted growth rates with increasing polynomial degree. Of course, the development of algorithms with optimal order complexity for the computation of the element matrices and load vectors are just one component of the finite element analysis, and in a practical implementation one must also take into account issues such as memory transfer and addressing.

Standard multi-index notations will be used throughout. In particular, for $\boldsymbol{\alpha} \in \mathbb{Z}^d_+$, we define $|\boldsymbol{\alpha}| = \sum_{k=1}^d \alpha_k$, $\boldsymbol{\alpha}! = \prod_{k=1}^d \alpha_k!$ and $\binom{|\boldsymbol{\alpha}|}{\boldsymbol{\alpha}} = |\boldsymbol{\alpha}|!/\boldsymbol{\alpha}!$. If $\boldsymbol{x} \in \mathbb{R}^d$, then we define $\boldsymbol{x}^{\boldsymbol{\alpha}} = \prod_{k=1}^d x_k^{\alpha_k}$. Given a pair $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^d_+$, $\boldsymbol{\beta} \leq \boldsymbol{\alpha}$ if and only if $\beta_k \leq \alpha_k$, $k = 1, \ldots, d$ and, in this case, $\binom{\boldsymbol{\alpha}}{\boldsymbol{\beta}} = \prod_{k=1}^d \binom{\alpha_k}{\beta_k}$.

2. Bernstein-Bézier Finite Elements

2.1. **Bernstein Polynomials.** Let $T = \mathrm{conv}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{d+1})$ be a non-degenerate simplex in $\mathbb{R}^d$. The set $\mathcal{D}^n_d(T) = \{\boldsymbol{x}_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{I}^n_d\}$ consists of the *domain points* of T defined by

$$\boldsymbol{x}_{\boldsymbol{\alpha}} = \frac{1}{n} \sum_{k=1}^{d+1} \alpha_k \boldsymbol{x}_k \tag{1}$$

where $\mathcal{I}^n_d$ is the indexing set

$$\mathcal{I}^n_d = \left\{ \boldsymbol{\alpha} \in \mathbb{Z}^{d+1}_+ : \ |\boldsymbol{\alpha}| = n \right\}. \tag{2}$$

The *barycentric coordinates* of a point $\boldsymbol{x} \in \mathbb{R}^d$ with respect to the simplex T are given by the unique $(d+1)$-tuple $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{d+1})$ satisfying

$$\boldsymbol{x} = \sum_{k=1}^{d+1} \lambda_k \boldsymbol{x}_k; \quad 1 = \sum_{k=1}^{d+1} \lambda_k. \tag{3}$$

For future reference, it is worth noting that

$$\operatorname{grad}\lambda_k = -\frac{|\gamma_k|}{d|T|}\boldsymbol{n}_k \tag{4}$$

where $\boldsymbol{n}_k$ is the unit outward normal on the face γ_k of the simplex T that does not have $\boldsymbol{x}_k$ as a vertex, and $|T|$ and $|\gamma_k|$ denote the d- and $(d-1)$-dimensional measures of the element and the face respectively. The Bernstein polynomials of degree $n \in \mathbb{Z}_+$ associated with T, defined by

$$B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \binom{n}{\boldsymbol{\alpha}}\boldsymbol{\lambda}^{\boldsymbol{\alpha}}, \quad \boldsymbol{\alpha} \in \mathcal{I}^n_d, \tag{5}$$

possess a number of remarkable properties [17], such as the de Casteljau algorithm for their efficient evaluation, that have led to their widespread use in the CAGD and computer graphics communities. It is apparent that the Bernstein polynomials are non-negative over the simplex T and, thanks to the multinomial theorem, sum to unity. Simple algebra suffices to see that the product of Bernstein polynomials is again a (scaled) Bernstein polynomial

$$B^m_{\boldsymbol{\alpha}}B^n_{\boldsymbol{\beta}} = \frac{\binom{\boldsymbol{\alpha}+\boldsymbol{\beta}}{\boldsymbol{\alpha}}}{\binom{m+n}{m}}B^{m+n}_{\boldsymbol{\alpha}+\boldsymbol{\beta}}, \quad \boldsymbol{\alpha} \in \mathcal{I}^m_d, \boldsymbol{\beta} \in \mathcal{I}^n_d, \tag{6}$$

whilst the integral of a Bernstein polynomial also has a simple form [17]:

$$\int_T B^n_{\boldsymbol{\alpha}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \frac{|T|}{\binom{n+d}{d}}, \quad \boldsymbol{\alpha} \in \mathcal{I}^n_d. \tag{7}$$

The Bernstein polynomials are linearly independent [17]. This, coupled with the observation that the cardinality of the indexing set $\mathcal{I}^n_d$ coincides with the dimension $\binom{n+\ell}{\ell}$ of the space $\mathbb{P}^n_d$, means that any polynomial $u \in \mathbb{P}^n_d$ has a unique *BB-form* representation

$$u = \sum_{\boldsymbol{\alpha}\in\mathcal{I}^n_d} c_{\boldsymbol{\alpha}}B^n_{\boldsymbol{\alpha}}. \tag{8}$$

The uniqueness of the *BB-vector*, $\{c_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{I}^n_d\}$, formed from the coefficients of the BB-form means that the set $\Sigma^n_d = \{\phi_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{I}^n_d\}$, consisting of linear functionals on $\mathbb{P}^n_d$ defined by the rules

$$\mathbb{P}^n_d \ni u \mapsto \phi_{\boldsymbol{\alpha}}(u) = c_{\boldsymbol{\alpha}}, \quad \boldsymbol{\alpha} \in \mathcal{I}^n_d, \tag{9}$$

is *unisolvent with respect to* $\mathbb{P}^n_d$, i.e.

$$u = 0 \quad \Longleftrightarrow \quad \phi_{\boldsymbol{\alpha}}(u) = 0 \text{ for all } \boldsymbol{\alpha} \in \mathcal{I}^n_d. \tag{10}$$

Consequently, the triple $(T, \Sigma^n_d, \mathbb{P}^n_d)$ is a finite element in the sense of [4, 7], which will be referred to as the *Bernstein-Bézier finite element (BB-FEM) of degree n on T*.

A *Lagrange finite element* would correspond to choosing $\phi_{\boldsymbol{\alpha}}(u)$ to be the value of the polynomial u at the domain point $\boldsymbol{x}_{\boldsymbol{\alpha}}$, establishing an obvious correspondence between domain points and degrees of freedom. Although the Bernstein-Bézier and Lagrange finite elements coincide in the case $n = 1$, the elements differ for higher orders $n > 1$. Occasionally, with a harmless abuse of nomenclature, it will be convenient to identify the domain point $\boldsymbol{x}_{\boldsymbol{\alpha}}$ with the local degree of freedom $\phi_{\boldsymbol{\alpha}}$.

2.2. **Bernstein-Bézier Finite Element Spaces on a Partition.** Let $\Delta = \{T\}$ be a regular partitioning (triangulation) of a bounded domain $\Omega \subset \mathbb{R}^d$ into the union of disjoint d-simplices T in the usual sense [4, 7]. In particular, the non-empty intersection $T \cap T'$ of any distinct pair T, $T' \in \Delta$ is an s-simplex in $\mathbb{R}^d$ formed by the common vertices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{s+1}$ of both T and T', so that

$$\mathcal{D}_s^n(T \cap T') = \mathcal{D}_d^n(T) \cap \mathcal{D}_d^n(T')$$

for some appropriate $0 \leq s \leq d$. This identity means that the domain points on neighbouring simplices match on the shared interface or, interpreted another way, that the local degrees of freedom are compatible between neighbouring elements.

Let u be a piecewise polynomial of degree n defined on the partition Δ, i.e. for each $T \in \Delta$, $u_{|T} = p_T \in \mathbb{P}_d^n(T)$. Then $p_T = p_{T'}$ on $T \cap T'$ if and only if the BB-vectors of p_T and $p_{T'}$ agree on the domain points $\mathcal{D}_s^n(T \cap T')$. In other words, if the values of the local degrees of freedom in the finite elements $(T, \Sigma_d^n(T), \mathbb{P}_d^n(T))$ and $(T', \Sigma_d^n(T'), \mathbb{P}_d^n(T'))$ agree on the common domain points, then the corresponding local functions p_T and $p_{T'}$ will be continuous across the interface between the elements, and vice versa. The same argument extends to intersections formed by three (or more) simplices belonging to Δ and, as a consequence, we obtain:

Theorem 1. *Let $S_d^n(\Delta)$ denote the finite element space defined over the partition Δ using Bernstein-Bézier finite elements $(T, \Sigma_d^n(T), \mathbb{P}_d^n(T))$, $T \in \Delta$. Then $S_d^n(\Delta)$ coincides with the standard (H^1-conforming) finite element space consisting of continuous piecewise polynomials of degree n on Δ: i.e.*

$$S_d^n(\Delta) = \left\{ u \in C^0(\Omega) : \; u|_T \in \mathbb{P}_d^n(T), \quad T \in \Delta \right\}. \tag{11}$$

The result shows that Bernstein polynomials may be used to construct a basis for the standard H^1-conforming finite element space consisting of continuous piecewise polynomials. Although Bernstein-Bézier representations have a number of properties which make them attractive for use in computationally demanding applications such CAGD and visualisation [11, 13, 14, 17], but the possibility of using them for finite element approximation has attracted virtually no attention amongst practitioners.

3. Evaluation of Bernstein-Bézier Moments

Let $T \in \Delta$ be a simplex and let $f : T \to X$ be a given smooth function, where X is a vector space (typically $\mathbb{R}$, $\mathbb{R}^d$ or $\mathbb{R}^{d \times d}$). The Bernstein-Bézier moments $\mu_{\boldsymbol{\alpha}}^n(f)$ of degree n for the function f on T are defined by:

$$\mu_{\boldsymbol{\alpha}}^n(f) = \int_T B_{\boldsymbol{\alpha}}^n(\boldsymbol{x}) f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \quad \boldsymbol{\alpha} \in \mathcal{I}_d^n. \tag{12}$$

In general, it is necessary to approximate these moments using an appropriate quadrature rule consisting of at least $\mathcal{O}(q^d)$ nodes, with $q \geq n$. If one were to simply employ the de Casteljau algorithm [17] to directly evaluate the Bernstein polynomial at each quadrature point at a cost of $\mathcal{O}(n^{d+1})$ operations for each index $\boldsymbol{\alpha} \in \mathcal{I}_d^n$, then the overall cost of evaluating the moments $\{\mu_{\boldsymbol{\alpha}}^n(f) : \boldsymbol{\alpha} \in \mathcal{I}_d^n\}$ would be $\mathcal{O}(n^{2d+1} q^d)$. Our objective in this section is to develop an algorithm whereby *all* of the moments can be evaluated at a cost of $\mathcal{O}(q^{d+1})$ operations.

Of course, if the function f is constant over the simplex T, then advantage may be taken of the closed form for the moments

$$\mu_{\boldsymbol{\alpha}}^n(f) = \frac{|T|}{\binom{n+d}{d}} f_{|T}, \quad \boldsymbol{\alpha} \in \mathcal{I}_d^n \tag{13}$$

which follows from (7).

3.1. **Bernstein-Bézier Bases and the Duffy Transformation.** The *Duffy transformation* [8, 9] applied to a point $\boldsymbol{t} = (t_1, t_2, \ldots, t_d) \in [0,1]^d$ may be defined recursively as follows:

$$\begin{aligned}
\lambda_1 &= t_1 \\
\lambda_2 &= t_2(1-\lambda_1) \\
\lambda_3 &= t_3(1-\lambda_1-\lambda_2) \\
&\vdots \\
\lambda_d &= t_d(1-\lambda_1-\lambda_2-\ldots-\lambda_{d-1}) \\
\lambda_{d+1} &= 1-\lambda_1-\lambda_2-\ldots-\lambda_d.
\end{aligned} \tag{14}$$

More generally, if $T = \operatorname{conv}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{d+1})$, then the Duffy transformation associated with the simplex T is defined by the rule

$$\boldsymbol{x}(\boldsymbol{t}) = \sum_{k=1}^{d+1} \lambda_k \boldsymbol{x}_k, \tag{15}$$

where λ_k are given by (14), and maps the unit cell $[0,1]^d$ onto the simplex T.

The tensor product structure can be exploited in a variety of ways, and helps account for the widespread usage of the Duffy transformation in conjunction with simplicial finite elements [15, 26]. The *Stroud conical quadrature rule* [23] is conveniently derived using this transformation to first express the integral of a function f over a simplex T in the form:

$$\int_T f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \frac{|T|}{d!}\int_0^1 \mathrm{d}t_1(1-t_1)^{d-1}\int_0^1 \mathrm{d}t_2(1-t_2)^{d-2}\cdots\int_0^1 \mathrm{d}t_d (f\circ\boldsymbol{x})(\boldsymbol{t}).$$

The q-point Gauss-Jacobi quadrature rule [23]:

$$\int_0^1 (1-s)^a s^b g(s)\,\mathrm{d}s \approx \sum_{j=1}^{q} \omega_j^{(a,b)} g(\xi_j^{(a,b)}) \tag{16}$$

has precision $2q-1$, the weights $\{\omega_j^{(a,b)}\}$ are all positive and the nodes $\{\xi_j^{(a,b)}\}$ are located on the interval $[0,1]$. The integral over the variable t_k is approximated using the Gauss-Jacobi rule with $a = d-k$ and $b = 0$, then we arrive at the *q-point Stroud rule* [23]:

$$\int_T f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{|T|}{d!}\sum_{i_1=1}^{q}\omega_{i_1}^{(d-1,0)}\sum_{i_2=1}^{q}\omega_{i_2}^{(d-2,0)}\cdots\sum_{i_d=1}^{q}\omega_{i_d}^{(0,0)} f(\boldsymbol{x}_{i_1,i_2,\ldots,i_d}), \tag{17}$$

which has positive weights, and consists of q^d nodes given by

$$\boldsymbol{x}_{i_1,i_2,\ldots,i_d} = \boldsymbol{x}(\xi_{i_1}^{(d-1,0)}, \xi_{i_2}^{(d-2,0)}\ldots, \xi_{i_d}^{(0,0)}) \tag{18}$$

for $1 \le i_1, i_2, \ldots, i_d \le q$. We shall make use of the Stroud quadrature rule later to approximate the Bernstein-Bézier moments.

An important observation, for our present purposes, is the favourable behaviour of the Bernstein polynomials under the Duffy transformation:

Lemma 1. *Let $B_k^m(t) = \binom{m}{k} t^k(1-t)^{m-k}$, $k \in \{0,1,\ldots,m\}$, denote the univariate Bernstein polynomials on the unit interval $[0,1]$. Then,*

$$B_{\boldsymbol{\alpha}}^n(\boldsymbol{x}(\boldsymbol{t})) = B_{\alpha_1}^n(t_1) B_{\alpha_2}^{n-\alpha_1}(t_2)\cdots B_{\alpha_d}^{n-\alpha_1-\ldots-\alpha_{d-1}}(t_d) \tag{19}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{d+1}) \in \mathcal{I}_d^n$ and $\boldsymbol{x}(\boldsymbol{t})$ denotes the Duffy transformation associated with T.

Proof. It is not difficult to show that (14) may be written in the form

$$\begin{aligned}
\lambda_1 &= t_1 \\
\lambda_2 &= t_2(1-t_1) \\
\lambda_3 &= t_3(1-t_1)(1-t_2) \\
&\vdots \\
\lambda_d &= t_d(1-t_1)(1-t_2)\dots(1-t_{d-1}) \\
\lambda_{d+1} &= (1-t_1)\dots(1-t_d).
\end{aligned}$$

Now,

$$B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}(\boldsymbol{t})) = \binom{n}{\boldsymbol{\alpha}} \lambda_1^{\alpha_1}\lambda_2^{\alpha_2}\dots\lambda_{d+1}^{\alpha_{d+1}}$$

and therefore, by inserting the above expressions for λ_k, collecting terms and simplifying and, on observing that

$$\binom{n}{\boldsymbol{\alpha}} = \binom{n}{\alpha_1}\binom{n-\alpha_1}{\alpha_2}\cdots\binom{n-\alpha_1-\ldots-\alpha_{d-1}}{\alpha_d},$$

we arrive at the claimed result. □

Property (19) is peculiar to the Bernstein-Bézier finite element and is not shared by other non-tensorial finite element bases. The Bernstein-Bézier finite element basis is rather unique in the respect that it is not based on a tensorial construction [8, 15, 26] but nevertheless possesses the key property of a tensorial basis needed to take advantage of the *sum factorisation* technique. The remainder of this section is concerned with describing how the technique may be exploited for the efficient implementation of the Bernstein-Bézier finite element.

3.2. **Application to Evaluation of Bernstein Polynomials.** We begin by illustrating how property (19) can be exploited for the efficient evaluation of a Bernstein polynomial of the form

$$u(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\mathcal{I}^n_d} c_{\boldsymbol{\alpha}} B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) \tag{20}$$

at all of the nodes $\boldsymbol{x}_{i_1,i_2,\ldots,i_d}$ of the Stroud conical quadrature rule given in equation (18). One approach would consist of applying the de Casteljau algorithm to evaluate u at each quadrature point at a cost of $\mathcal{O}(n^{d+1})$ operations, giving an overall cost of $\mathcal{O}(n^{d+1}q^d)$ operations to evaluate at all quadrature points. Fortunately, there is an efficient alternative.

Changing variable using the Duffy transformation and using property (19) gives the alternative form

$$(u\circ\boldsymbol{x})(\boldsymbol{t}) = \sum_{\alpha_1=0}^{n} B^n_{\alpha_1}(t_1) \sum_{\alpha_2=0}^{n-\alpha_1} B^{n-\alpha_1}_{\alpha_2}(t_2)\dots \sum_{\alpha_d=0}^{n-\alpha_1-\ldots-\alpha_{d-1}} B^{n-\alpha_1-\ldots-\alpha_{d-1}}_{\alpha_d}(t_d) c_{\boldsymbol{\alpha}}.$$

Algorithm 1: Evaluate(C^0, q)

Input: Array C^0 corresponding to BB-vector $\{c_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{I}_d^n\}$ of a polynomial u, and index q for number of points.
Output: Array C^d with entry $[i_1, \ldots, i_d]$ equal to value of u at Stroud node.
for $\ell = d$ **to** 1 **do**
  $C^{d-\ell+1} = \mathsf{EvalStep}(C^{d-\ell}, \ell, q)$;
return C^d;

Algorithm 2: EvalStep(C^{in}, ℓ, q)

Input: Array C^{in} and index $\ell \in \{1, \ldots, d\}$.
Output: Updated array C^{out} with index α_ℓ switched to i_ℓ.
for $i_\ell = 1$ **to** q **do**
  $\xi = \xi_{i_\ell}^{(d-\ell,\,0)}$; $s = 1 - \xi$; $r = \xi/s$;
  foreach $(\alpha_1, \ldots, \alpha_{\ell-1}, \bullet) \in \mathcal{I}_{\ell-1}^n$ **do**
    $w = s^{n-\alpha_1-\ldots-\alpha_{\ell-1}}$;
    for $\alpha_\ell = 0$ **to** $n - \alpha_1 - \ldots - \alpha_{\ell-1}$ **do**
      foreach $(i_{\ell+1}, \ldots, i_d) \in \{1, \ldots, q\}^{d-\ell}$ **do**
        //w given by $B_{\alpha_\ell}^{n-\alpha_1-\ldots-\alpha_{\ell-1}}(\xi_{i_\ell}^{(d-\ell,0)})$ here.
        $C^{out}[\alpha_1, \ldots, \alpha_{\ell-1}, i_\ell, i_{\ell+1}, \ldots, i_d]$
        $\mathrel{+}= w * C^{in}[\alpha_1, \ldots, \alpha_{\ell-1}, \alpha_\ell, i_{\ell+1} \ldots, i_d]$;
      $w \mathrel{*}= r * (n - \alpha_1 - \ldots - \alpha_\ell)/(1 + \alpha_\ell)$;
return C^{out};

An algorithm for computing the values of u at the Stroud nodes efficiently can be developed based on expressing this identity in recursive form:

$$\begin{aligned}
C^0(\alpha_1, \ldots, \alpha_{d-1}, \alpha_d) = {} & c_{\alpha_1,\ldots,\alpha_{d-1},\alpha_d} \\
C^1(\alpha_1, \ldots, \alpha_{d-1}, i_d) = {} & \sum_{\alpha_d=0}^{n-\alpha_1-\ldots-\alpha_{d-1}} B_{\alpha_d}^{n-\alpha_1-\ldots-\alpha_{d-1}}(\xi_{i_d}^{(0,0)}) C^0(\alpha_1, \alpha_2, \ldots, \alpha_d) \\
C^2(\alpha_1, \ldots, i_{d-1}, i_d) = {} & \sum_{\alpha_{d-1}=0}^{n-\alpha_1-\ldots-\alpha_{d-2}} B_{\alpha_{d-1}}^{n-\alpha_1-\ldots-\alpha_{d-2}}(\xi_{i_{d-1}}^{(1,0)}) C^1(\alpha_1, \ldots, \alpha_{d-1}, i_d) \\
& \vdots \\
C^d(i_1, \ldots, i_{d-1}, i_d) = {} & \sum_{\alpha_1=0}^{n} B_{\alpha_1}^{n}(\xi_{i_d}^{(d-1,0)}) C^{d-1}(\alpha_1, \ldots, i_{d-1}, i_d)
\end{aligned} \tag{21}$$

where the indices in the ℓ-th step range over $\alpha_1, \alpha_2, \ldots, \alpha_{d-\ell}$ such that $(\alpha_1, \ldots, \alpha_{d-\ell}, \bullet) \in \mathcal{I}_{d-\ell}^n$ and $1 \le i_{d-\ell+1}, \ldots, i_d \le q$. The values of the Bernstein polynomial at the Stroud nodes are then given by the components of the final array C^d produced by (21):

$$u(\boldsymbol{x}_{i_1,\ldots,i_{d-1},i_d}) = C^d(i_1, \ldots, i_{d-1}, i_d). \tag{22}$$

The approach suggested by the recursion relations (21) is sometimes described as the *sum factorisation procedure* [10, 15, 19].

The routine Evaluate defined in Algorithm 1 evaluates by applying the procedure EvalStep defined in Algorithm 2 recursively to the BB-vector for u:

Theorem 2. *Let u be the Bernstein polynomial (20) and let C be the corresponding BB-vector $\{c_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{I}_d^n\}$. Let U denote the array given by $U = \mathsf{Evaluate}(C, q)$. Then,*

$$u(\boldsymbol{x}_{i_1,\ldots,i_d}) = U[i_1, \ldots, i_d] \tag{23}$$

for $1 \le i_1, \ldots, i_d \le q$. Moreover, the number of operations needed to compute U is of order $\mathcal{O}(q^{d+1}(e^{(n+1)/q} - 1))$.

Proof. We begin by considering the effect of a single application of algorithm $\mathsf{EvalStep}(C^{in}, \ell)$ on an array C^{in} for $\ell \in \{1, ..., d\}$. Observe that for given indices $(\alpha_1, \ldots, \alpha_{\ell-1}, t) \in \mathcal{I}_\ell^n$ and $i_\ell \in \{1, \ldots, q\}$, the local variable w appearing in the listing of $\mathsf{EvalStep}$ is initially set to be $w = s^{n-\alpha_1-\ldots-\alpha_{\ell-1}}$, where $s = 1 - \xi_{i_\ell}^{(d-\ell,0)}$, and then updated on each passage through the loop over $\alpha_\ell = 0, \ldots, n - \alpha_1 - \ldots - \alpha_{\ell-1}$ by multiplying through by a factor $r(n - \alpha_1 - \ldots - \alpha_\ell)/(1 + \alpha_\ell)$. We claim that the value of w used to update C^{out} when the loop index is α_ℓ is given by $B_{\alpha_\ell}^{n-\alpha_1-\ldots-\alpha_{\ell-1}}(\xi_{i_\ell}^{(d-\ell,0)})$. This can be seen by noting that the Bernstein polynomials satisfy the recurrence relation

$$B_0^m(t) = (1-t)^m; \quad B_{k+1}^m(t) = r\frac{m-k}{1+k}B_k^m(t), \quad k = 0, 1, \ldots, m-1 \tag{24}$$

where $r = t/(1-t)$, choosing $m = n - \alpha_1 - \ldots - \alpha_{\ell-1}$, $k = \alpha_\ell$ and $t = \xi_{i_\ell}^{(d-\ell,0)}$, and using induction.

Algorithm $\mathsf{EvalStep}$ updates all entries $[\alpha_1, \ldots, \alpha_{\ell-1}, i_\ell, \ldots, i_d]$ of the output array C^{out}, satisfying $(\alpha_1, \ldots, \alpha_{\ell-1}, \bullet) \in \mathcal{I}_\ell^n$ and $1 \le i_\ell, \ldots, i_d \le q$, with the sum over $\alpha_\ell \in \{0, \ldots, n-\alpha_1-\ldots-\alpha_{\ell-1}\}$ of the quantities $w * C^{in}[\alpha_1, \ldots, \alpha_{\ell-1}, \alpha_\ell, i_{\ell+1} \ldots, i_d]$ where, as shown above, $w = B_{\alpha_\ell}^{n-\alpha_1-\ldots-\alpha_{\ell-1}}(\xi_{i_\ell}^{(d-\ell,0)})$. It follows that C^{out} is computed in terms of C^{in} by the same expression as the one used to obtain $C^{d-\ell+1}$ from $C^{d-\ell}$ in the recurrence relations (21):

$$C^{out}[\alpha_1, \ldots, \alpha_{\ell-1}, i_\ell, i_{\ell+1}, \ldots, i_d] = \sum_{\alpha_\ell=0}^{n-\alpha_1-\ldots-\alpha_{\ell-1}} B_{\alpha_\ell}^{n-\alpha_1-\ldots-\alpha_{\ell-1}}(\xi_{i_\ell}^{(d-\ell,0)}) C^{in}[\alpha_1, \ldots, \alpha_{\ell-1}, \alpha_\ell, i_{\ell+1}, \ldots, i_d].$$

This, coupled with the fact that C^0 is chosen to be the coefficient vector for the polynomial u in Bernstein-Bézier form, means that the resulting array C^d indeed gives the values of the polynomial u at the Stroud points.

Finally, a single application of $\mathsf{EvalStep}$ with index ℓ consists of q outer loops over i_ℓ, each of which requires $\dim \mathcal{I}_{\ell-1}^n$ loops over the index $\alpha_1, \ldots, \alpha_{\ell-1}$ and $n - \alpha_1 - \ldots - \alpha_{\ell-1}$ loops over α_ℓ, whilst the innermost update of C^{out} requires $q^{d-\ell}$ loops over the indices $i_\ell, \ldots, i_d$. The complexity is dominated by the innermost loop which is executed $q^{d-\ell+1} \cdot \dim \mathbb{P}_\ell^n$ times, each time requiring one addition and one multiplication operation. The total complexity of computing C^d is therefore of order

$$q^{d+1} \sum_{\ell=1}^{d} \binom{n+\ell}{\ell} q^{-\ell}.$$

By considering the Taylor polynomial of degree d for the function $(1-x)^{-(n+1)}$, evaluated at $x = 1/q$, and using Taylor's Theorem, it is not difficult to show that

$$\left(\frac{q}{q-1}\right)^{n+1} \left(1 - \mathcal{O}(q^{-(d+1)})\right) \le \sum_{\ell=0}^{d} \binom{n+\ell}{\ell} q^{-\ell} \le \left(\frac{q}{q-1}\right)^{n+1}.$$

Making use of the estimate

$$e^{(n+1)/q} \leq \left(\frac{q}{q-1}\right)^{n+1} \leq e^{(n+1)/(q-1)}.$$

gives the claimed result. □

Algorithm EvalStep makes use of the recurrence relation (24) for evaluating the univariate Bernstein polynomials, which becomes unstable when the argument is close to unity. However, one could modify EvalStep to exploit the backward recurrence relation

$$B_m^m(t) = t^m; \quad B_{k-1}^m(t) = \frac{1-t}{t}\frac{k}{m-k+1}B_k^m(t), \quad k = m, \ldots, 1 \tag{25}$$

if the argument $t > 1/2$, whilst using forward recurrence (24) if $t \leq 1/2$.

Typically, the number of quadrature points q is chosen to be the degree n of the polynomial (or a small multiple thereof). Theorem 2 shows that one can compute the values of a degree n polynomial at all of the Stroud points in just $\mathcal{O}(q^{d+1})$ operations: i.e. *the algorithm evaluates u at all q^d Stroud points with the same complexity as using the de Casteljau algorithm to evaluate at a single point.* Of course, the points at which we are evaluating u will not form a regular subdivision of the simplex, although efficient algorithms are available [20] to deal with that case.

The sum factorisation method is generally used in conjunction with pre-computed values of the univariate basis functions at quadrature points to develop efficient finite element procedures [10, 15]. The analogue in our case would consist of pre-computing the values of the univariate Bernstein polynomials at the Gauss-Jacobi points and dispensing with the local variable w in EvalStep in favour of directly accessing a stored value of the basis function. However, the overall complexity would not be improved beyond that already attained using the algorithm presented in Theorem 2 which *does not require* pre-computing, or perhaps more importantly *accessing* of, stored values of basis functions. The latter point is important given that memory access on some kinds of hardware can be orders of magnitude slower than the cost of performing floating point operations.

One is often interested in derivatives of a polynomial u written in Bernstein-Bézier form (20). For example, on using the chain rule and (4), we find that

$$\operatorname{grad} u = -\frac{1}{d|T|}\sum_{k=1}^{d+1}|\gamma_k|\frac{\partial u}{\partial \lambda_k}\boldsymbol{n}_k \tag{26}$$

where the partial derivatives are taken regarding $\lambda_1, \ldots, \lambda_{d+1}$ as independent variables. Likewise, higher order derivatives can be expressed as a linear combination of unconstrained partial derivatives of u with respect to the variables representing the barycentric coordinates. For a given multi-index $\boldsymbol{\nu}$ with $|\boldsymbol{\nu}| \leq n$, the $\boldsymbol{\nu}$-th derivative of u may be expressed in Bernstein-Bézier form:

$$\frac{\partial^{|\boldsymbol{\nu}|}u}{\partial \lambda^{\boldsymbol{\nu}}} = \frac{n!}{(n-|\boldsymbol{\nu}|)!}\sum_{\boldsymbol{\alpha}\in\mathcal{I}_d^{n-|\boldsymbol{\nu}|}} c_{\boldsymbol{\alpha}+\boldsymbol{\nu}}B_{\boldsymbol{\alpha}}^{n-|\boldsymbol{\nu}|}. \tag{27}$$

The derivative is a Bernstein polynomial of degree $n-|\boldsymbol{\nu}|$ with BB-vector given by

$$c_{\boldsymbol{\alpha}}^{\boldsymbol{\nu}} = \frac{n!}{(n-|\boldsymbol{\nu}|)!}c_{\boldsymbol{\alpha}+\boldsymbol{\nu}}, \quad \boldsymbol{\alpha}\in\mathcal{I}_d^{n-|\boldsymbol{\nu}|}.$$

For convenience, if C denotes BB-vector of u, then we shall denote the BB-vector of the derivative $\partial_{\boldsymbol{\nu}}u$ using the shorthand notation

$$\partial_{\boldsymbol{\nu}}C = \{n!c_{\boldsymbol{\alpha}+\boldsymbol{\nu}}/(n-|\boldsymbol{\nu}|)! : \boldsymbol{\alpha}\in\mathcal{I}_d^{n-|\boldsymbol{\nu}|}\}. \tag{28}$$

Of course, if $|\boldsymbol{\nu}|$ exceeds the order n of the polynomial, then the derivative is trivial.

Corollary 1. *Let u denote the Bernstein polynomial (20) and C be the corresponding BB-vector. Given $\boldsymbol{\nu} \in \mathcal{I}_d^\ell$, $0 \le \ell \le n$, let $U_{\boldsymbol{\nu}}$ denote the array given by $U_{\boldsymbol{\nu}} = \mathsf{Evaluate}(\partial_{\boldsymbol{\nu}} C, q)$. Then,*

$$\frac{\partial^{|\boldsymbol{\nu}|} u}{\partial \lambda^{\boldsymbol{\nu}}}(\boldsymbol{x}_{i_1,\ldots,i_d}) = U_{\boldsymbol{\nu}}[i_1,\ldots,i_d] \tag{29}$$

for $1 \le i_1,\ldots,i_d \le q$. Moreover, the number of operations required to compute $U_{\boldsymbol{\nu}}$ is of order $\mathcal{O}(q^d(e^{(n+1-\ell)/q}-1))$.

3.3. **Application to Evaluation of Bernstein-Bézier Moments.** The transformation property (19) can be similarly exploited in the computation of the Bernstein-Bézier moments $\{\mu^n_{\boldsymbol{\alpha}}(f) : \boldsymbol{\alpha} \in \mathcal{I}^n_d\}$. The starting point is again a change of variable using the Duffy transformation in conjunction with property (19) to give

$$\begin{aligned}
\mu^n_{\boldsymbol{\alpha}}(f) = \quad & \frac{|T|}{d!}\int_0^1 \mathrm{d}t_d B^{n-\alpha_1-\ldots-\alpha_{d-1}}_{\alpha_d}(t_d) \\
& \times \int_0^1 \mathrm{d}t_{d-1}(1-t_{d-1}) B^{n-\alpha_1-\ldots-\alpha_{d-2}}_{\alpha_{d-1}}(t_{d-1}) \\
& \cdots \\
& \times \int_0^1 \mathrm{d}t_2 (1-t_2)^{d-2} B^{n-\alpha_1}_{\alpha_2}(t_2) \\
& \times \int_0^1 \mathrm{d}t_1 (1-t_1)^{d-1} B^n_{\alpha_1}(t_1)(f\circ\boldsymbol{x})(\boldsymbol{t})
\end{aligned} \tag{30}$$

which leads to the following recursion relations

$$\begin{aligned}
F^0(t_1,t_2,\ldots,t_d) = \quad & (f\circ\boldsymbol{x})(\boldsymbol{t}) \\
F^1(\alpha_1,t_2,\ldots,t_d) = \quad & \int_0^1 \mathrm{d}t_1 (1-t_1)^{d-1} B^n_{\alpha_1}(t_1) F^0(t_1,t_2,\ldots,t_d) \\
F^2(\alpha_1,\alpha_2,\ldots,t_d) = \quad & \int_0^1 \mathrm{d}t_2 (1-t_2)^{d-2} B^{n-\alpha_1}_{\alpha_2}(t_2) F^1(\alpha_1,t_2,\ldots,t_d) \\
& \vdots \\
F^d(\alpha_1,\alpha_2,\ldots,\alpha_d) = \quad & \int_0^1 \mathrm{d}t_d B^{n-\alpha_1-\ldots-\alpha_{d-1}}_{\alpha_d}(t_d) F^{d-1}(\alpha_1,\alpha_2,\ldots,t_d),
\end{aligned} \tag{31}$$

with the moments given by

$$\mu^n_{\boldsymbol{\alpha}}(f) = \frac{|T|}{d!} F^d(\alpha_1,\alpha_2,\ldots,\alpha_d). \tag{32}$$

We evaluate the integrals appearing in (31) using the Stroud conical quadrature rule (17), which amounts to replacing the integral defining F^k in (31) by the Gauss-Jacobi rule (16) with weights $a = d-k$ and $b = 0$. This leads to the following recursion relations for computing the approximate moments $\mu^n_{\boldsymbol{\alpha}}(f)$ (we shall not distinguish between true quantities and their approximations using the quadrature

Algorithm 3: Moment(F^0, q)

Input: Array F^0 corresponding to values of a function f at Stroud nodes.
Output: Array F^d composed of Bernstein-Bézier moments of f using Stroud rule.

```
for ℓ = 1 to d do
    F^ℓ = MomentStep(F^{ℓ−1}, ℓ, q);
return F^d;
```

Algorithm 4: MomentStep(F^{in}, ℓ, q)

Input: Array F^{in} and index $\ell \in \{1, \dots, d\}$.
Output: Updated array F^{out} with index i_ℓ switched to α_ℓ.

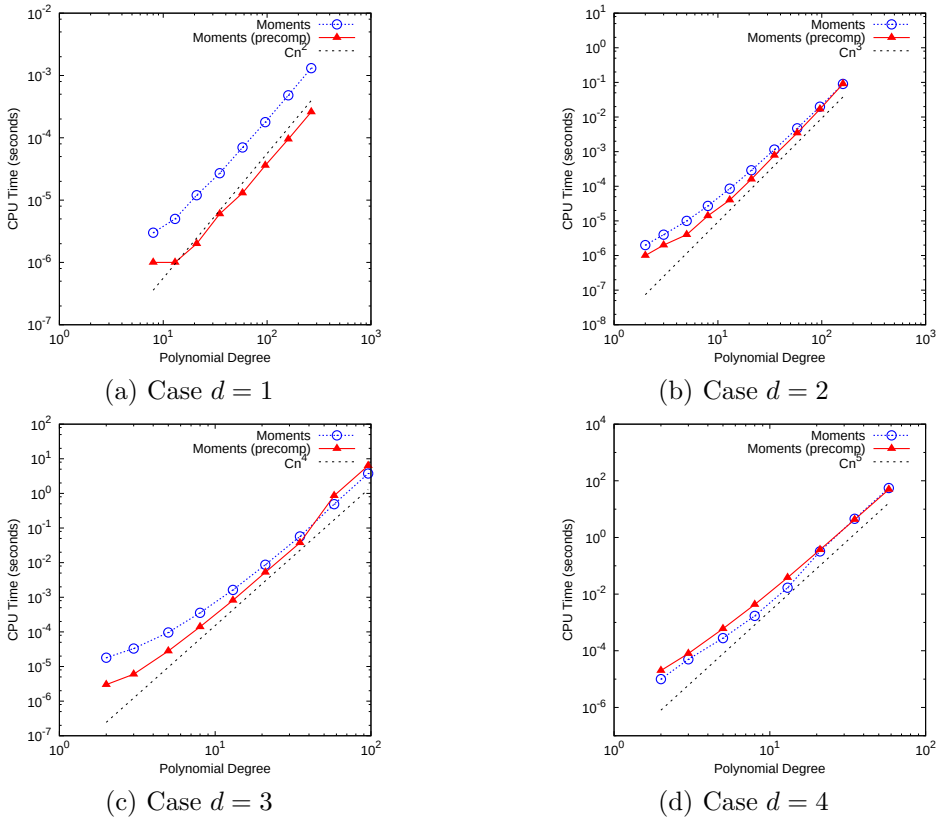
```
for i_ℓ = 1 to q do
    ξ = ξ_{i_ℓ}^{(d−ℓ, 0)}; ω = ω_{i_ℓ}^{(d−ℓ, 0)}; s = 1 − ξ; r = ξ/s;
    foreach (α_1, ..., α_{ℓ−1}, •) ∈ I^n_{ℓ−1} do
        w = ω ∗ s^{n−α_1−...−α_{ℓ−1}};
        for α_ℓ = 0 to n − α_1 − ... − α_{ℓ−1} do
            foreach (i_{ℓ+1}, ..., i_d) ∈ {1, ..., q}^{d−ℓ} do
                //w given by ω_{i_ℓ}^{(d−ℓ, 0)} B^{n−α_1−...−α_{ℓ−1}}_{α_ℓ}(ξ_{i_ℓ}^{(d−ℓ,0)}) here.
                F^{out}[α_1, ..., α_{ℓ−1}, α_ℓ, i_{ℓ+1}, ..., i_d]
                  += w ∗ F^{in}[α_1, ..., α_{ℓ−1}, i_ℓ, i_{ℓ+1} ..., i_d];
            w ∗= r ∗ (n − α_1 − ... − α_ℓ)/(1 + α_ℓ);
return F^{out};
```

rules):

$$
\begin{aligned}
F^0(i_1, i_2, \dots, i_d) = \quad & (f \circ \boldsymbol{x})(\xi_{i_1}^{(d-1,0)}, \xi_{i_2}^{(d-2,0)}, \dots, \xi_{i_d}^{(0,0)}) \\
F^1(\alpha_1, i_2, \dots, i_d) = \quad & \sum_{i_1=1}^{q} \omega_{i_1}^{(d-1,0)} B_{\alpha_1}^{n}(\xi_{i_1}^{(d-1,0)}) F^0(i_1, i_2, \dots, i_d) \\
F^2(\alpha_1, \alpha_2, \dots, i_d) = \quad & \sum_{i_2=1}^{q} \omega_{i_2}^{(d-2,0)} B_{\alpha_2}^{n-\alpha_1}(\xi_{i_2}^{(d-2,0)}) F^1(\alpha_1, i_2, \dots, i_d) \\
& \vdots \\
F^d(\alpha_1, \alpha_2, \dots, \alpha_d) = \quad & \sum_{i_d=1}^{q} \omega_{i_d}^{(0,0)} B_{\alpha_d}^{n-\alpha_1-\dots-\alpha_{d-1}}(\xi_{i_d}^{(0,0)}) F^{d-1}(\alpha_1, \alpha_2, \dots, i_d)
\end{aligned}
\tag{33}
$$

where the indices in the k-th step range over the values $0 \le \alpha_1,\ \alpha_2,\ \dots,\ \alpha_k \le n$ subject to $\alpha_1 + \alpha_2 + \dots + \alpha_k \le n$, and $1 \le i_{k+1},\ \dots,\ i_d \le n$. The non-negativity of the Bernstein polynomials along with the weights of the Gauss-Jacobi quadrature rules means that the quadrature rule, viewed as being applied to the data f, is positive.

The system (33) has obvious similarities with the system (21) used in the evaluation of a Bernstein polynomial. The main differences are that summation over α_ℓ is replaced by summation over i_ℓ, and that the quadrature weight $\omega_{i_\ell}^{(d-\ell,0)}$ appears as an additional factor in the summand. The procedure MomentStep defined in Algorithm 4 reflects these differences but shares strong similarities with EvalStep.



(a) Case $d = 1$ (b) Case $d = 2$

(c) Case $d = 3$ (d) Case $d = 4$

FIGURE 1. CPU time required for computation of moments using Algorithm 3 compared with same approach using precomputed arrays of values of basis functions at quadrature points.

We have the following consequence of Theorem 2:

Corollary 2. *Let T be a simplex in d-dimensions and $f : T \to X$ be a smooth function, with X a vector space. Let F be the array corresponding to the values of f at the Stroud nodes on the simplex T (c.f. the first equation of (33)), and define $M = \mathsf{Moment}(F, q)$. Then, the Bernstein-Bézier moments of f over T, approximated using the Stroud rule, are given by*

$$\mu^n_{\boldsymbol{\alpha}}(f) = \frac{|T|}{d!} M[\alpha_1, \alpha_2, \ldots, \alpha_d], \quad \boldsymbol{\alpha} \in \mathcal{I}^n_d. \tag{34}$$

Moreover, the number of operations needed to compute M is of order $q^{d+1}(e^{(n+1)/q} - 1)\mathsf{Op}(X)$, where $\mathsf{Op}(X)$ is the cost of performing the operation $y \mathrel{+}= c * x$, for given vectors $x, y \in X$ and a scalar c.

Proof. Virtually identical to that of Theorem 2 after observing that the innermost loop of MomentStep is executed the same number of times as in EvalStep and costs $\mathsf{Op}(X)$. □

The computation of the values of the data f at the Stroud points can be accomplished using an algorithm based on (14) which involves $\mathcal{O}(q^d)$ operations and q^d function evaluations. We shall leave the precise details as an (easy) exercise.

Figure 1 shows the growth of the CPU time required to compute the moment vector with the polynomial degree using Algorithm 3 (obtained using a Dell Precision T7400 workstation with Xeon 3.2GHz processor and 32Gb RAM, using C++

and the `gcc` compiler (version 4.1.2)). The number of quadrature points q is taken to be twice the polynomial degree n. The CPU time grows as $\mathcal{O}(n^{d+1})$ as predicted by Corollary 2. For comparison, results are presented in the case when the univariate Bernstein-Bézier functions are evaluated on the fly (as in Algorithm 4) and when the values at the quadrature points are precomputed and stored on disc. It is observed that the difference in CPU time is negligible.

3.4. **Evaluation of Moments in the Non-Linear Case.** Let $f : T\times\mathbb{R}\times\mathbb{R}^d \to X$ be a given smooth function, where X is again a vector space. The finite element approximation of non-linear problems often entails the computation of the moments $\mu^n_{\boldsymbol{\alpha}}(u, f)$, $\boldsymbol{\alpha} \in \mathcal{I}^n_d$, with respect to the current finite element iterate u: e.g.

$$\mu^n_{\boldsymbol{\alpha}}(u, f) = \int_T B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) f(\boldsymbol{x}; u(\boldsymbol{x}); \operatorname{grad} u(\boldsymbol{x}))\, \mathrm{d}\boldsymbol{x}. \tag{35}$$

The current finite element iteration u is expressed in Bernstein-Bézier form (20) and as a consequence, by composing the procedures described above, we can construct an efficient procedure to evaluate the non-linear moments:

Corollary 3. *Let $f : T \times \mathbb{R} \times \mathbb{R}^d \to X$ be a smooth function, where X is again a vector space and u be a Bernstein polynomial (20) with BB-vector C. Let M denote the array computed as follows:*

- $U = \mathsf{Evaluate}(C, q)$*;*
- $U_{\boldsymbol{\nu}} = \mathsf{Evaluate}(\partial_{\boldsymbol{\nu}} C, q)$, $\boldsymbol{\nu} \in \mathcal{I}^1_d$ *and compute* $\operatorname{grad} U$ *using (26);*
- *set* $F[i_1, \ldots, i_d] = f(\boldsymbol{x}_{i_1,\ldots,i_d}; U[i_1, \ldots, i_d]; \operatorname{grad} U[i_1, \ldots, i_d])$*;*
- $M = \mathsf{Moment}(F, q)$.

Then, M contains the Stroud approximation of the non-linear Bernstein-Bézier moments $\mu^n_{\boldsymbol{\alpha}}(u, f)$, $\boldsymbol{\alpha} \in \mathcal{I}^n_d$, and the process needed to compute M requires $\mathcal{O}(q^{d+1}(e^{(n+1)/q} - 1))\mathsf{Op}(X)$ operations.

The evaluation of moments involving higher order derivatives of the polynomial u could be achieved in the same complexity by extending the second step to compute the relevant higher derivatives.

The algorithms presented in this section show that (the Stroud approximations of) the Bernstein-Bézier moments can be computed efficiently, even in the case of non-linear problems where the integrand depends on the current finite element approximation. Furthermore, the moments are available in closed form in the case of piecewise constant data over a triangulation.

4. Optimal Order Element Level Computations

Let $\Omega \subset \mathbb{R}^d$ be a polyhedral domain with boundary $\partial\Omega = \Gamma_D \cap \Gamma_N$, where Γ_D and Γ_N are disjoint. Consider the model problem of finding $u \in H^1_D(\Omega)$ such that

$$B(u; u, v) = L(u; v) \quad \forall v \in H^1_D(\Omega) \tag{36}$$

Algorithm 5: Multinomial$(\boldsymbol{A}, m, n)$

Input: Precomputed binomial coefficients $\{C_p^{p+q} : 0 \le p \le m, 0 \le q \le n\}$.

Output: $\boldsymbol{A}$ such that $\boldsymbol{A}_{\boldsymbol{\alpha}\boldsymbol{\beta}} = \binom{\boldsymbol{\alpha}+\boldsymbol{\beta}}{\boldsymbol{\alpha}} / \binom{m+n}{m}$, $\boldsymbol{\alpha} \in \mathcal{I}_d^m$, $\boldsymbol{\beta} \in \mathcal{I}_d^n$. Note: Sums appearing in loop conditions should be stored, and not recomputed (as shown here for simplicity).

```
for α_1 = 0 to m do
    for β_1 = 0 to n do
        w_1 = C_{α_1}^{α_1+β_1} / C_m^{m+n};
        for α_2 = 0 to m − α_1 do
            for β_2 = 0 to n − β_1 do
                w_2 = w_1 * C_{α_2}^{α_2+β_2};
                  ⋮
                for α_d = 0 to m − α_1 − ... − α_{d−1} do
                    for β_d = 0 to n − β_1 − ... − β_{d−1} do
                        w_d = w_{d−1} * C_{α_d}^{α_d+β_d};
                        w_{d+1} = w_d * C_{m−α_1−...−α_d}^{m−α_1−...−α_d+n−β_1−...−β_d};
>                       A_{αβ} = w_{d+1};
                    ⋮
```

where $H_D^1(\Omega)$ is the usual Sobolev space with vanishing traces on Γ_D and the forms B and L are given by

$$\begin{aligned} B(w; u, v) &= \int_\Omega \operatorname{grad} v \cdot \boldsymbol{A}(\boldsymbol{x}, w(\boldsymbol{x}), \operatorname{grad} w(\boldsymbol{x})) \cdot \operatorname{grad} u \\ &+ \int_\Omega v\boldsymbol{b}(\boldsymbol{x}, w(\boldsymbol{x}), \operatorname{grad} w(\boldsymbol{x})) \cdot \operatorname{grad} u \\ &+ \int_\Omega c(\boldsymbol{x}, w(\boldsymbol{x}), \operatorname{grad} w(\boldsymbol{x}))uv \\ &+ \int_{\Gamma_N} c_N(\boldsymbol{x}, w(\boldsymbol{x}))uv \,\mathrm{d}s \end{aligned}$$

and

$$L(w; v) = \int_\Omega f(\boldsymbol{x}, w(\boldsymbol{x}), \operatorname{grad} w(\boldsymbol{x}))v \,\mathrm{d}\boldsymbol{x} + \int_{\Gamma_N} f_N(\boldsymbol{x}, w(\boldsymbol{x}))v \,\mathrm{d}s.$$

We shall not dwell on formulating conditions on the data under which the above problem is well-posed but shall assume this to be the case. We note that the above formulation is sufficiently general as to encompass a very broad class of non-linear partial differential equations.

One of the major bottlenecks in finite element codes, particularly in the case of higher order elements, lies in the assembly of the global finite element system. This entails the computation of element level contributions to the global load vector $\boldsymbol{f}$, given (in the linear case) by

$$\boldsymbol{f}_{\boldsymbol{\alpha}}^T = \int_T f(\boldsymbol{x}) B_{\boldsymbol{\alpha}}^n(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}, \quad \boldsymbol{\alpha} \in \mathcal{I}_d^n \tag{37}$$

and element level contributions to the global system matrix $\boldsymbol{B}$, given (in the linear case) by

$$\boldsymbol{B}_{\boldsymbol{\alpha}\boldsymbol{\beta}}^T = \int_T \left(\operatorname{grad} B_{\boldsymbol{\beta}}^n \cdot \boldsymbol{A}(\boldsymbol{x}) \operatorname{grad} B_{\boldsymbol{\alpha}}^n + B_{\boldsymbol{\beta}}^n \boldsymbol{b}(\boldsymbol{x}) \cdot \operatorname{grad} B_{\boldsymbol{\alpha}}^n + c(\boldsymbol{x}) B_{\boldsymbol{\beta}}^n B_{\boldsymbol{\alpha}}^n \right) \mathrm{d}\boldsymbol{x} \tag{38}$$

Algorithm 6: MassMat($\boldsymbol{M}, n, \{\mu^{2n}_{\boldsymbol{\alpha}}(c), \boldsymbol{\alpha} \in \mathcal{I}^{2n}_d\}$)

Input: Bernstein-Bézier moments $\{\mu^{2n}_{\boldsymbol{\alpha}}(c) : \boldsymbol{\alpha} \in \mathcal{I}^{2n}_d\}$ computed with $q = n + 1$ quadrature points.
Output: Element mass matrix $\boldsymbol{M}$.

Code as in Multinomial($\boldsymbol{M}$, n, n) with the line
> $\boldsymbol{A}_{\boldsymbol{\alpha\beta}} = w_{d+1}$;
replaced with the line
> $\boldsymbol{M}_{\boldsymbol{\alpha\beta}} = w_{d+1} * \mu^{2n}_{\boldsymbol{\alpha+\beta}}(c)$;

for $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \in \mathcal{I}^n_d$.

The element load vector contains $\binom{n+d}{n}$ entries which coincide with the Bernstein-Bézer moments of the data f:

$$\boldsymbol{f}^T_{\boldsymbol{\alpha}} = \int_T f(\boldsymbol{x}) B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = \mu^n_{\boldsymbol{\alpha}}(f), \quad \boldsymbol{\alpha} \in \mathcal{I}^n_d. \tag{39}$$

The moments can be approximated using the Stroud conical product rule with $q = n + 1$ points in $\mathcal{O}(n^{d+1})$ operations thanks to Corollary 2. Moreover, the same conclusion holds also in the non-linear case using the approach described in Corollary 3.

The element matrix $\boldsymbol{B}^T$ contains $\binom{n+d}{n}^2$ entries which means that (even if each entry could be computed in $\mathcal{O}(1)$ operations) the overall cost of computing the element matrix is at least $\mathcal{O}(n^{2d})$. It is clear that the element stiffness matrix is the bottleneck in the assembly of the finite element system.

The algorithms presented in this section enable the element matrix for the Bernstein-Bézier finite element to be assembled in complexity $\mathcal{O}(n^{2d})$ operations. Thus, the algorithms attain optimal complexity for the assembly of the element matrix. Naturally, the contributions from elements belonging to the (lower dimensional) boundary Γ_N can be obtained using exactly the same techniques at a complexity of $\mathcal{O}(n^{2d-2})$ operations. The main idea consists of exploiting the results of Corollaries 2 and 3 for the efficient computation of the Bernstein-Bézier moments of the data. These results apply to the case of both linear and non-linear partial differential equations. However, for ease of notation, we shall present the results for the linear case. It is nevertheless worth emphasising that all of the conclusions extend to the general non-linear case, with the only difference being that the non-linear moments are computed using the procedure of Corollary 3 instead of Corollary 2.

4.1. **Mass Matrix.** The element mass matrix $\boldsymbol{M}^T$ for degree n polynomials in $\mathbb{R}^d$ has dimension $\binom{n+d}{d} \times \binom{n+d}{d}$, with entries given by

$$\boldsymbol{M}^T_{\boldsymbol{\alpha\beta}} = \int_T c(\boldsymbol{x}) B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) B^n_{\boldsymbol{\beta}}(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}, \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}^n_d. \tag{40}$$

The entries are approximated using the Stroud conical rule based on $q = n + 1$ points. One of the consequences of the identity (6) is that the entries in the mass matrix can be written in terms of the Bernstein-Bézier moments of c of order $2n$,

$$\boldsymbol{M}^T_{\boldsymbol{\alpha\beta}} = \frac{\binom{\boldsymbol{\alpha+\beta}}{\boldsymbol{\alpha}}}{\binom{2n}{n}} \mu^{2n}_{\boldsymbol{\alpha+\beta}}(c), \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}^n_d. \tag{41}$$

This observation forms the basis for an optimal order algorithm which enables the element mass matrix to be constructed in $\mathcal{O}(n^{2d})$ operations but requires an appropriate algorithm for the evaluation of the multinomial coefficients. A simple

Algorithm 7: StiffMat($\boldsymbol{S}, n, \{\mu_{\boldsymbol{\alpha}}^{2n-2}(\boldsymbol{A}), \boldsymbol{\alpha} \in \mathcal{I}_d^{2n-2}\}$)

Input: Bernstein-Bézier moments $\{\mu_{\boldsymbol{\alpha}}^{2n-2}(\boldsymbol{A}) : \boldsymbol{\alpha} \in \mathcal{I}_d^{2n-2}\}$ computed with $q = n+1$ quadrature points.
Output: Element stiffness matrix $\boldsymbol{S}$.

Compute
> **for** $k = 1$ **to** $d+1$ **do**
> **for** $\ell = 1$ **to** $d+1$ **do**
> $\tilde{\mu}_{\boldsymbol{\alpha}}^{(k,\ell)} = \operatorname{grad}\lambda_\ell \cdot \mu_{\boldsymbol{\alpha}}^{2n-2}(\boldsymbol{A}) \cdot \operatorname{grad}\lambda_k, \quad \boldsymbol{\alpha} \in \mathcal{I}_d^{2n-2}$;

Remaining code as in Multinomial($\boldsymbol{S}$, $n-1$, $n-1$) with the line
> $\boldsymbol{A}_{\boldsymbol{\alpha}\boldsymbol{\beta}} = w_{d+1}$;
replaced with the lines
> **for** $k = 1$ **to** $d+1$ **do**
> **for** $\ell = 1$ **to** $d+1$ **do**
> $\boldsymbol{S}_{\boldsymbol{\alpha}+\boldsymbol{e}_k,\boldsymbol{\beta}+\boldsymbol{e}_\ell}$ += $n^2 * w_{d+1} * \tilde{\mu}_{\boldsymbol{\alpha}+\boldsymbol{\beta}}^{(k,\ell)}$;

algorithm is presented in Algorithm 5, although we note that the treatment of expressions involving large multinomial coefficients is typical when dealing with Bernstein polynomials and requires some care [12] to avoid overflow whilst maintaining efficiency.

Theorem 3. *The element mass matrix of degree n in $\mathbb{R}^d$ can be approximated and assembled using the Stroud conical quadrature rule with $q = n+1$ points in $\mathcal{O}(n^{2d})$ operations using Algorithm* MassMat.

Proof. Thanks to Corollary 2, the Bernstein-Bézier moments of order $2n$ based on the Stroud conical quadrature rule with $q = n+1$ points, can be evaluated in $\mathcal{O}(n^{d+1}(e^{(2n+1)/n} - 1))$ operations. It is easy to verify that Algorithm 5 results in $w_{d+1} = \binom{\boldsymbol{\alpha}+\boldsymbol{\beta}}{\boldsymbol{\alpha}} / \binom{2n}{n}$ and hence, thanks to (41), MassMat assembles the element mass matrix from the moments. For $\ell = 1, \ldots, d-1$, the loop over the pair of indices α_ℓ, β_ℓ contains one multiplication or division and is executed a total of $\binom{n+\ell}{\ell}^2$ times, whilst the innermost loop, $\ell = d$, requires three multiplications and is executed $\binom{n+d}{d}^2$ times, resulting in an overall complexity of

$$\sum_{\ell=1}^{d} \binom{n+\ell}{\ell}^2 + 2\binom{n+d}{d}^2 = \mathcal{O}(n^{2d})$$

operations, as claimed. □

4.2. **Stiffness Matrix.** The element stiffness matrix $\boldsymbol{S}^T$ for degree n polynomials in $\mathbb{R}^d$ has dimension $\binom{n+d}{d} \times \binom{n+d}{d}$, with entries given by

$$\boldsymbol{S}^T_{\boldsymbol{\alpha}\boldsymbol{\beta}} = \int_T \operatorname{grad} B^n_{\boldsymbol{\beta}}(\boldsymbol{x}) \cdot \boldsymbol{A}(\boldsymbol{x}) \cdot \operatorname{grad} B^n_{\boldsymbol{\alpha}}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}, \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}_d^n \tag{42}$$

where the gradient is taken with respect to the physical coordinate $\boldsymbol{x} \in T$. The main idea again is to exploit the property (6) of the Bernstein polynomials. Let $\boldsymbol{e}_\ell \in \mathcal{I}_d^1$ denote the multi-index whose ℓ-th entry is unity and whose remaining entries vanish. We shall first use the following identity (easily obtained using elementary arguments and the definition of the Bernstein polynomials)

$$\operatorname{grad} B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) = n \sum_{k=1}^{d+1} B^{n-1}_{\boldsymbol{\alpha}-\boldsymbol{e}_k}(\boldsymbol{x}) \operatorname{grad}\lambda_k, \quad \boldsymbol{\alpha} \in \mathcal{I}_d^n \tag{43}$$

Algorithm 8: ConvectMat($\boldsymbol{V}, n, \{\mu^{2n-1}_{\boldsymbol{\alpha}}(\boldsymbol{b}), \boldsymbol{\alpha} \in \mathcal{I}^{2n-1}_d\}$)

Input: Bernstein-Bézier moments $\{\mu^{2n-1}_{\boldsymbol{\alpha}}(\boldsymbol{b}) : \boldsymbol{\alpha} \in \mathcal{I}^{2n-1}_d\}$ computed with $q = n+1$ quadrature points.
Output: Element convective matrix $\boldsymbol{V}$.

Compute
> **for** $k = 1$ **to** $d+1$ **do**
> $\quad \tilde{\mu}^{(k)}_{\boldsymbol{\alpha}} = \mu^{2n-2}_{\boldsymbol{\alpha}}(\boldsymbol{b}) \cdot \operatorname{grad} \lambda_k, \quad \boldsymbol{\alpha} \in \mathcal{I}^{2n-1}_d;$

Code as in Multinomial($\boldsymbol{V}$, $n-1$, n) with the line
> $\boldsymbol{A}_{\boldsymbol{\alpha\beta}} = w_{d+1};$
replaced with the lines
> **for** $k = 1$ **to** $d+1$ **do**
> $\quad \boldsymbol{V}_{\boldsymbol{\alpha}+\boldsymbol{e}_k,\boldsymbol{\beta}}$ += $n * w_{d+1} * \tilde{\mu}^{(k)}_{\boldsymbol{\alpha}+\boldsymbol{\beta}};$

where we use the convention whereby any terms for which $\boldsymbol{\alpha} - \boldsymbol{e}_k \notin \mathcal{I}^{n-1}_d$ are simply omitted from the summation. Inserting this identity into the expression for the entries of the stiffness and making use of the property (6) gives an expression for the entries of the stiffness matrix in terms of the matrix-valued Bernstein-Bézier moments of the data $\boldsymbol{A} \in \mathbb{R}^{d\times d}$:

$$\boldsymbol{S}^T_{\boldsymbol{\alpha\beta}} = n^2 \sum_{k,\ell=1}^{d+1} \frac{\binom{\boldsymbol{\alpha}-\boldsymbol{e}_k+\boldsymbol{\beta}-\boldsymbol{e}_\ell}{\boldsymbol{\alpha}-\boldsymbol{e}_k}}{\binom{2n-2}{n-1}} \operatorname{grad} \lambda_k \cdot \mu^{2n-2}_{\boldsymbol{\alpha}-\boldsymbol{e}_k+\boldsymbol{\beta}-\boldsymbol{e}_\ell}(\boldsymbol{A}) \cdot \operatorname{grad} \lambda_\ell \tag{44}$$

for $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}^n_d$, where we again adopt the convention whereby terms for which $\boldsymbol{\alpha}-\boldsymbol{e}_k+\boldsymbol{\beta}-\boldsymbol{e}_\ell \notin \mathcal{I}^{2n-2}_d$ are ignored in the summation. Algorithm 7 takes advantage of the expression (44) for the efficient assembly of the element stiffness matrix in a similar fashion to the case of the mass matrix.

Theorem 4. *The element stiffness matrix of degree n in $\mathbb{R}^d$ can be approximated and assembled using the Stroud conical quadrature rule with $q = n+1$ points in $\mathcal{O}(n^{2d})$ operations using Algorithm* StiffMat.

The proof is virtually identical to the case of the mass matrix. One point worth noting however, is that the algorithm implements the convention (whereby terms involving invalid indices are ignored) without requiring expensive conditional **if** statements.

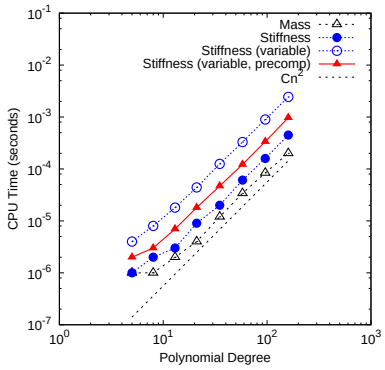
4.3. **Convective Matrix.** The element matrix $\boldsymbol{V}^T$ corresponding to the convective term is given by

$$\boldsymbol{V}^T_{\boldsymbol{\alpha\beta}} = \int_T B^n_{\boldsymbol{\beta}}(\boldsymbol{x}) \boldsymbol{b}(\boldsymbol{x}) \cdot \operatorname{grad} B^n_{\boldsymbol{\alpha}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}^n_d. \tag{45}$$

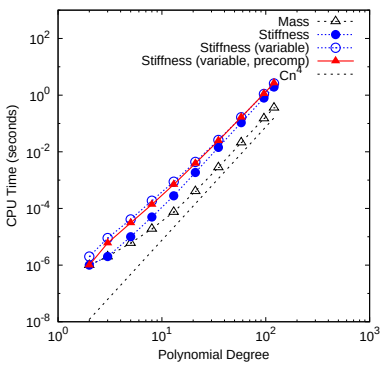
Arguing as in the case of the element stiffness matrix using identities (6) and (43) leads to an alternative expression for these entries in terms of the vector-valued Bernstein-Bézier moments of the data $\boldsymbol{b} \in \mathbb{R}^d$:

$$\boldsymbol{V}^T_{\boldsymbol{\alpha\beta}} = n \sum_{k=1}^{d+1} \frac{\binom{\boldsymbol{\alpha}-\boldsymbol{e}_k+\boldsymbol{\beta}}{\boldsymbol{\alpha}-\boldsymbol{e}_k}}{\binom{2n-1}{n-1}} \mu^{2n-1}_{\boldsymbol{\alpha}-\boldsymbol{e}_k+\boldsymbol{\beta}}(\boldsymbol{b}) \cdot \operatorname{grad} \lambda_k \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{I}^n_d, \tag{46}$$

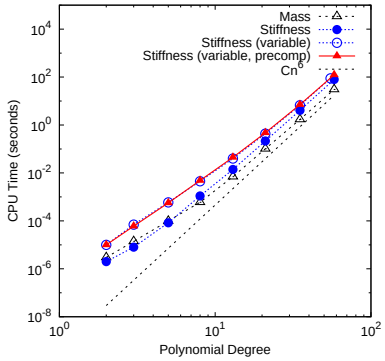
with the usual convention applying. Algorithm 8 exploits this expression for the efficient assembly of the element convective matrix:



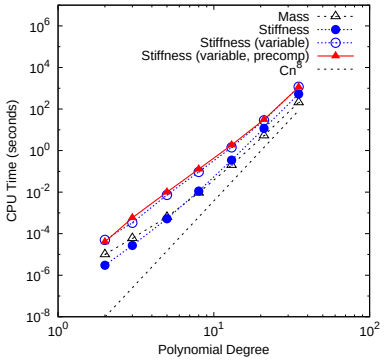
(a) Case $d = 1$



(b) Case $d = 2$



(a) Case $d = 3$



(b) Case $d = 4$

FIGURE 2. CPU time required for assembly of stiffness and mass matrices using Algorithms 6 and 7. The moments are obtained directly from (13) in the case of constant data, and evaluated using Algorithm 3 in the case of variable data. Results are shown for the cases of dynamic computation and precomputed values of the basis functions at the quadrature points.

Theorem 5. *The element convective matrix of degree n in $\mathbb{R}^d$ can be approximated and assembled using the Stroud conical quadrature rule with $q = n + 1$ points in $\mathcal{O}(n^{2d})$ operations using Algorithm* ConvectMat.

Figure 2 presents the timings obtained in the case where the number of quadrature points q is taken to be $n+1$, where n denotes the polynomial degree. The CPU time grows as $\mathcal{O}(n^{2d})$ as predicted by theorems 3 and 4. For comparison, results are presented in the cases when the coefficients $\boldsymbol{A}$ and c are piecewise constant and when they vary within the element. It is observed that there is some benefit in taking advantage of the closed form expression (13) in the case of piecewise constant data although the difference diminishes as the order n increases in the cases where $d > 1$, reflecting the fact that the computation of the moments is an $\mathcal{O}(n^{d+1})$ operation compared with the actual assembly which is an $\mathcal{O}(n^{2d})$ operation. Results are presented for variable data for when the univariate Bernstein-Bézier functions are evaluated on the fly (as in Algorithm 4) and for when the values at the quadrature points are precomputed and stored on disc. It is observed that the difference in CPU time is negligible. The actual implementation takes advantage of sequential lay-out of the entries in the matrices in memory to avoid excessive costs due to indirect memory access.

4.4. **Efficient Multiplication by Element Matrices.** The algorithms presented above show how the computation of the element matrices can be carried out (using the Stroud conical quadrature rule) for polynomials of arbitrary degree n in any number of dimensions d in the optimal complexity of $\mathcal{O}(n^{2d})$ operations. Nevertheless, in some applications, such as iterative solution methods, one may wish to by-pass the assembly of the matrices in favour of directly computing the result of multiplication of the BB-vector $\vec{C}$ corresponding to the current approximation u to the true solution by one of the above matrices.

By choosing $f(\boldsymbol{x}; u; \operatorname{grad} u) = c(\boldsymbol{x})u(\boldsymbol{x})$ in equation (35), we obtain

$$\mu^n_\alpha(u, f) = \int_T B^n_\alpha(\boldsymbol{x})c(\boldsymbol{x})u(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \sum_{\beta\in\mathcal{I}^n_d} \boldsymbol{M}_{\alpha\beta}C_\beta = (\boldsymbol{M}\vec{C})_\alpha, \quad \alpha \in \mathcal{I}^n_d,$$

and it follows from Corollary 3 that multiplication by the mass matrix can be carried out in $\mathcal{O}(q^{d+1}(e^{(n+1)/q} - 1))$ operations, even in the case of variable data. Likewise, choosing $f(\boldsymbol{x}; u; \operatorname{grad} u) = \boldsymbol{A}(\boldsymbol{x}) \operatorname{grad} u$ in equation (35) and making use of identity (43) enables us to compute the result of multiplication by the stiffness matrix in $\mathcal{O}(q^{d+1}(e^{(n+1)/q} - 1))$ operations, again in the case of variable data. If, as is usually the case, we select $q = \mathcal{O}(n)$, then these operation counts reduce to $\mathcal{O}(n^{d+1})$ complexity obtained in [16] in the special case of piecewise constant data.

5. Tensor Product Bernstein-Bézier Finite Elements

Suppose that we have a finite element (A, Σ_A, P_A) where P_A is the space spanned by the basis functions $\{\phi^A_\alpha : \alpha \in \Sigma_A\}$, and that the valuation of the moments $\{\mu^\alpha_A(\cdot) : \alpha \in \Sigma_A\}$ costs M_A operations and involves N_A function evaluations. Given another such element (B, Σ_B, P_B), the tensor product element over the domain $A \times B$ has degrees of freedom $\Sigma_A \times \Sigma_B$ over the space $P_A \times P_B$ spanned by $\{\phi^A_\alpha(x)\phi^B_\beta(y) : \alpha \in \Sigma_A, \beta \in \Sigma_B)$.

How much does it cost to evaluate the moments

$$\mu^{\alpha\times\beta}_{A\times B}(f) = \int_{A\times B} f(x, y)\phi^A_\alpha(x)\phi^B_\beta(y)\,\mathrm{d}x\,\mathrm{d}y, \quad (\alpha, \beta) \in \Sigma_A \times \Sigma_B$$

of a given function $f : A \times B \to \mathbb{R}$, for the tensor product element? It is easy to see that

$$\mu^{\alpha\times\beta}_{A\times B}(f) = \mu^\alpha_A(f^\beta_B), \quad \alpha \in \Sigma_A, \beta \in \Sigma_B$$

where $f^\beta_B : A \to \mathbb{R}$, $\beta \in \Sigma_B$, is the function defined by the rule

$$f^\beta_B(x) = \int_B f(x, y)\phi^B_\beta(y)\,\mathrm{d}y = \mu^\beta_B(f(x, \cdot)).$$

The computation of the moments over the tensor product element requires N_A evaluations of the function f^β_B for each $\beta \in \Sigma_B$, thereby contributing a cost of $N_A \times M_B$ operations. Once these values are in hand, the cost of evaluating the moments μ^α_A for each $\beta \in \Sigma_B$ is M_A, meaning that the computation of the actual moments given the data contributes a cost of $M_A \times \dim(\Sigma_B)$. It is clear that the number of function evaluations N_B must exceed the dimension of the space P_B, so that $\dim(\Sigma_B) \le N_B$. The following result now follows:

Theorem 6. *Let (A, Σ_A, P_A), respectively (B, Σ_B, P_B), denote a finite element whose moments may be computed in M_A operations and N_A function evaluations, respectively M_B and N_B. Then, the moments for the tensor product element may be evaluated in at most $M_A \times N_B + N_A \times M_B$ operations involving $N_A \times N_B$ function evaluations.*

We may use the above construction and result to construct other kinds of Bernstein-Bézier finite elements and corresponding algorithms for the evaluation of the moments.

5.1. **Example 1: Prismatic Bernstein-Bézier Finite Element.** Suppose that we form a tensor product element from a d-dimensional and a 1-dimensional simplex giving a so-called prismatic element in the case $d \geq 2$. We suppose that each element has the same polynomial degree n and each is based on the same q-point Stroud rule, i.e. $(A, \Sigma_A, P_A) = (T_d, \Sigma_d^n, \mathbb{P}_d^n)$ and $(B, \Sigma_B, P_B) = (T_1, \Sigma_1^n, \mathbb{P}_1^n)$. Then, the tensor product element uses Bernstein-Bézier basis functions. Moreover, thanks to Corollary 2, we have $N_A = q^d$, $M_A = \mathcal{O}(q^{d+1}(e^{(n+1)/q} - 1))$ and $N_B = q$, $M_B = \mathcal{O}(q^2(e^{(n+1)/q} - 1))$. Theorem 6 then shows that the moments on the tensor product element can be computed in at most $\mathcal{O}(q^{d+2}(e^{(n+1)/q} - 1))$ operations by composing the routine Moment with itself to first compute the moments over B, and then again to compute the moments over A. Consequently, we can evaluate the moments over the $d + 1$-dimensional tensor product element with the same complexity as needed to evaluate the moments over a $d + 1$-dimensional simplex.

5.2. **Example 2: Quadrilateral Bernstein-Bézier Finite Element.** Suppose we take $d = 1$ in the previous example. Then the resulting element is a Bernstein-Bézier quadrilateral finite element and the moments can be evaluated in $\mathcal{O}(q^2(n+1))$ operations involving q^2 function evaluations since $M_A = M_B = \mathcal{O}(q(n + 1))$ in this case (c.f. proof of Theorem 2).

5.3. **Example 3: Hexahedral Bernstein-Bézier Finite Element.** Suppose we take the element A appearing in Theorem 6 to be the quadrilateral Bernstein-Bézier finite element in the previous example and B to be the Bernstein-Bézier finite element $(T_1, \Sigma_1^n, \mathbb{P}_1^n)$. The resulting element is a Bernstein-Bézier hexahedral finite element and the moments can be evaluated in $\mathcal{O}(q^3(n + 1))$ operations and q^3 function evaluations.

5.4. **Example 4: Bernstein-Bézier Finite Element on $[0, 1]^d$.** More generally, by repeatedly forming the tensor product of this element with $(T_1, \Sigma_1^n, \mathbb{P}_1^n)$, we obtain the d-dimensional tensor product Bernstein-Bézier finite element for which the moments can be evaluated in $\mathcal{O}(q^d(n + 1))$ operations.

The foregoing examples show how Bernstein-Bézier finite elements can be constructed on elements other than simplices and that the moments can be evaluated in the same complexity as the corresponding simplicial element in the same number of dimensions. The algorithms for the assembly of the stiffness, mass and convective matrices are readily extended to tensor product elements to give optimal order assembly algorithms for such elements.

References

[1] S. Adjerid, M. Aiffa, and J. Flaherty, *Hierarchical finite element bases for triangular and tetrahedral elements*, Comput. Methods Appl. Mech. Engrg., 190 (2001), pp. 2925–2941.

[2] M. Ainsworth and J. Coyle, *Hierarchic finite element bases on unstructured tetrahedral meshes*, Internat. J. Numer. Methods Engrg., 58 (2003), pp. 2103–2130.

[3] M. L. Bittencourt, *Fully tensorial nodal and modal shape functions for triangles and tetrahedra*, Internat. J. Numer. Methods Engrg., 63 (2005), pp. 1530–1558.

[4] S. Brenner and L. Scott, *The Mathematical Theory of Finite Element Methods*, vol. 15 of Texts in Applied Mathematics, Springer-Verlag, New York, 1994.

[5] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral methods. Evolution to complex geometries and applications to fluid dynamics*, Scientific Computation, Springer, Berlin, 2007.

[6] P. Carnevali, R. Morris, Y. Tsuji, and G. Taylor, *New basis functions and computational procedures for p-version finite element analysis*, Internat. J. Numer. Methods Engrg., 35 (1993), pp. 3759–3779.
[7] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, Elsevier, North-Holland, 1978.
[8] M. Dubiner, *Spectral methods on triangles and other domains*, J. Sci. Comput., 6 (1991), pp. 345–390.
[9] M. G. Duffy, *Quadrature over a pyramid or cube of integrands with a singularity at a vertex*, SIAM J. Numer. Anal., 19 (1982), pp. 1260–1262.
[10] T. Eibner and J. Melenk, *Fast algorithms for setting up the stiffness matrix in hp-fem: a comparison*, in Computer Mathematics and its Applications: Advances and Developments (1994–2005), E. A. Lipitakis, ed., Athens, Greece, 2006, LEA Publishers, pp. 575–596.
[11] G. Farin, *Curves and surfaces for CAGD: a practical guide, Fifth Edition*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
[12] R. T. Farouki and V. T. Rajan, *Algorithms for polynomials in Bernstein form*, Comput. Aided Geom. Design, 5 (1988), pp. 1–26.
[13] J. Foley, A. van Dan, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, Inc., 1996.
[14] J. Hoschek and D. Lasser, *Fundamentals of computer aided geometric design*, A K Peters Ltd., Wellesley, MA, 1993. Translated from the 1992 German edition by Larry L. Schumaker.
[15] G. E. Karniadakis and S. J. Sherwin, *Spectral/hp element methods for computational fluid dynamics*, Numerical Mathematics and Scientific Computation, Oxford University Press, New York, second ed., 2005.
[16] R. Kirby, *Fast simplicial finite element algorithms using bernstein polynomials*, Numer. Math., (2010), pp. 1–22. 10.1007/s00211-010-0327-2.
[17] M.-J. Lai and L. L. Schumaker, *Spline functions on triangulations*, vol. 110 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, 2007.
[18] J. M. Melenk, K. Gerdes, and C. Schwab, *Fully discrete hp-finite elements: fast quadrature*, Comput. Methods Appl. Mech. Engrg., 190 (2001), pp. 4339 – 4364.
[19] S. A. Orszag, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92.
[20] J. Peters, *Evaluation and approximate evaluation of the multivariate Bernstein-Bézier form on a regularly partitioned simplex*, ACM Trans. Math. Software, 20 (1994), pp. 460–480.
[21] L. Schumaker, *Computing bivariate splines in scattered data fitting and the finite element method*, Numer. Alg., 48 (2008), pp. 237–260.
[22] C. Schwab, *p- and hp-Finite Element Methods: Theory and Applications in Solid and Fluid Mechanics*, Numerical Mathematics and Scientific Computation, Oxford University Press, 1998.
[23] A. H. Stroud, *Approximate calculation of multiple integrals*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1971. Prentice-Hall Series in Automatic Computation.
[24] B. Szabo and I. Babuska, *Finite Element Analysis*, John Wiley & Sons, 1991.
[25] P. E. J. Vos, S. J. Sherwin, and R. M. Kirby, *From h to p efficiently: implementing finite and spectral/hp element methods to achieve optimal performance for low- and high-order discretisations*, J. Comput. Phys., 229 (2010), pp. 5161–5181.
[26] T. C. Warburton, S. J. Sherwin, and G. E. Karniadakis, *Basis functions for triangular and quadrilateral high-order elements*, SIAM J. Sci. Comp., 20 (1999), pp. 1671–1695 (electronic).

Mark Ainsworth, Gaelle Andriamaro, Oleg Davydov, Department of Mathematics and Statistics, Strathclyde University, 26 Richmond St., Glasgow G1 1XH, Scotland.
E-mail address: {M.Ainsworth, G.Andriamaro, O.Davydov}@strath.ac.uk