**Numerische Mathematik manuscript No.**
(will be inserted by the editor)

# Fast simplicial quadrature-based finite element operators using Bernstein polynomials

**Robert C. Kirby**[1] ⋆**, Kieu Tri Thinh**[1]

Texas Tech University

Received: date / Revised version: date

**Summary** We derive low-complexity matrix-free finite element algorithms for simplicial Bernstein polynomials on simplices. Our techniques, based on a sparse representation of differentiation and special block structure in the matrices evaluating B-form polynomials at warped Gauss points, apply to variable coefficient problems as well as constant coefficient ones, thus extending our results in [14].

**Key words** finite element, fast algorithm, Bernstein polynomials

---

*Send offprint requests to*:

⋆ *Present address:* Texas Tech University; Department of Mathematics and Statistics; PO Box 1042; Lubbock, TX 79409-1042. Work supported by the National Science Foundation under award number 0830655.

## 1 Introduction

In [14], we study constant coefficient finite element operators in simplicial geometry with Bernstein polynomials as local bases. We derived matrix-free algorithms for applying the local finite element operators with complexity comparable to that of tensor-product techniques used for spectral element methods in rectangular domains. These techniques relied on recognizing special structure in the constant coefficient mass matrix, so it was left as an open question how to find fast quadrature-based algorithms for more general variational forms. In this paper, we develop such algorithms by exposing structure in certain higher-dimensional Bernstein-Vandermonde matrices that represent polynomial evaluation and finite element integation using warped Gauss point integration rules. These techniques, more subtle than those based on coordinating Gauss-Lobatto quadrature and interpolation nodes for tensor-product bases [8,9], demonstrate that variable coefficient operators may in fact be applied with low complexity.

In particular, we are focused on bilinear forms of the type

$$b(u,v) = \int_\Omega f(x) D_1 u D_2 v \, dx,$$

where $D_1$ and $D_2$ are some differential operators (including the identity). In finite element calculations, such bilinear forms are typically

evaluated by meshing $\Omega$ into simple shapes, such as simplices, and evaluating the restriction of the bilinear form over such shapes. We let $\mathcal{T}$ denote a triangulation of $\Omega$ [7]. Then, we are concerned with

$$b_T(u, v) = \int_T f(x) D_1 u D_2 v \, dx,$$

where $T \in \mathcal{T}$ is a single simplex in a finite element mesh. The function $f$ may either be a specified function of space, or may depend on some function in a finite element space. Let $\{\psi_i\}_{i=1}^N$ be a local basis for the finite element space on $T$, such as the set of Bernstein polynomials, this allows us to define a matrix

$$K_{T,ij} = \int_T f D_1 \psi_j D_2 \psi_i \, dx. \tag{1}$$

Then, if $u = \sum_{i=1}^N u_i \psi_i$ is a vector expressed in the local basis, the action of $K$ on $u$ is given by

$$(K_T u)_i = \int_T f(x) D_1 u D_2 \psi_i \, dx. \tag{2}$$

It is possible to evaluate the entries of $K_T u$ given in (2) by numerical quadrature without forming $K$ explicitly. If $\{(x_i, w_i)\}_{i=1}^M$ denotes a set of quadrature weights and points, a basic matrix-free algorithm is given in Algorithm 1. This is a quadratic process, for if $N_0 = \max(M, N)$, then $\mathcal{O}(N_0^2)$ operations per cell are required, supposing that a matrix-vector product is used to tabulate $D_1 u$ at the quadrature points.

**Algorithm 1** Matrix-free evaluation of $y = Ku$

$y \leftarrow 0$

Let $\mu_i = D_1 u$ evaluated at $x_i$.

Let $V_{ij}$ be $D_2 \psi_j$ evaluated at $x_i$.

**for** $j \leftarrow 1, N$ **do**

  **for** $i \leftarrow 1, M$ **do**

    $y_j \leftarrow y_j + w_i V_{ij} \mu_i$

  **end for**

**end for**

As a major cost of matrix-based methods is the formation and storage of the global system matrix, using matrix-free algorithm can be attractive. The global finite element matrix can be expressed as

$$K = \mathcal{A}^t K_{\mathcal{T}} \mathcal{A}, \tag{3}$$

where $\mathcal{A}$ is a sparse matrix that expresses the restriction of global basis functions in terms of the basis on each cell $T$, and $K_{\mathcal{T}}$ is a block diagonal matrix matrix whose blocks consist of the elementwise matrices (1) for each $T \in \mathcal{T}$. When $\mathcal{A}$ is used to explicitly construct a global matrix, the process is typically called *assembly* and requires explicit values for each $K_T$. Otherwise, only the action of $K_T$, by algorithm 1 or otherwise, on a vector is required.

While using Algorithm 1 cell-by-cell has a comparable arithmetic cost as using a global matrix, it requires considerably less mem-

ory. However, both techniques become quite expensive for high-order methods, as the cost grows quadratically in the size of the local polynomial basis, which in terms depends on a power of the polynomial degree.

Fast algorithms for polynomial evaluation and integration are common in rectangular domains, where a tensor product structure is readily apparent. On simplices, Karniadakis and Sherwin [12] construct special bases for which fast algorithms are available. Our present interest is in a "generic" finite element basis, the Bernstein polynomials, showing that they admit fast algorithms as well. Having an extremely flexible basis such as the Bernstein polynomials will allow future work extending these techniques to exterior calculus bases developed by Arnold, Falk, and Winther [2,3] as well as spline-based methods [5,4, 16,18,11]. Also, while this paper was under review, similar techniques were independtly discovered by Ainsworth, Andriamaro, and Davydov [**?**]. Their approach obtains similar algorithms by means of the Duffy transform acting on integrals rather than our linear algebraic approach, and they also provide an efficient algorithm for constructing entire elementwise matrices.

In the remainder of the paper, we shall develop necessary notation for Bernstein polynomials and give a formulation of the warped Gauss

rules in barycentric coordinates in Section 2. Our techniques for evaluation and integration will be based on Bernstein-Vandermonde matrices that evaluate Bernstein polynomials at these particular quadrature rules. We will derive explicit formulas for these matrices in terms of the one-dimensional Gauss quadrature points for one-, two-, and three-dimensional simplices in Section 3. Evaluating a polynomial at the warped Gauss-points is equivalent to multiplying its Bernstein expansion coefficients by these Bernstein-Vandermonde matrices, and we derive reduced-complexity algorithms for two and three dimensions in Section 4. Forming the necessary finite element vectors (2) by numerical integration requires the transpose of these matrices, and we will use slightly different techniques to obtain the action of the transpose in Section 5.

## 2 Notation and preliminaries

### *2.1 Barycentric coordinates*

For integer $d \geq 1$, let $S_d$ be a nondegenerate simplex in $\mathbb{R}^d$. Let $\{x_i\}_{i=1}^{d+1} \subset \mathbb{R}^d$ be the vertices of $S_d$. Let

$$\{b_i\}_{i=1}^{d+1}$$

denote the *barycentric coordinates* of $S_d$. Each $b_i$ is an affine map from $\mathbb{R}^d$ to $\mathbb{R}$ such that

$$b_i(x_j) = \delta_{ij}$$

for each vertex $x_j$, where $\delta_{ij}$ is the standard Kronecker delta. Each $b_i$ is nonnegative on $S_d$, and

$$\sum_{i=1}^{d+1} b_i = 1.$$

*2.2 Bernstein polynomials*

A multiindex of length $d$ is a $d$-tuple of nonnegative integers, written

$$\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_d).$$

The *order* of a multiindex $\alpha$, written $|\alpha|$ is given by the sum of its components. That is,

$$|\alpha| = \sum_{i=1}^{d} \alpha_i.$$

If $\alpha, \beta$ have length $d$, their sum is defined componentwise by

$$\alpha + \beta = (\alpha_1 + \beta_1, \alpha_2 + \beta_2, \cdots, \alpha_d + \beta_d).$$

The factorial of a multiindex is defined as the product of the components' factorials by

$$\alpha! = \Pi_{i=1}^{d} \alpha_i!$$

Let $\mathbf{b}_d = (b_1, b_2, \cdots, b_{d+1})$ denote the tuple of barycentric coordinates on a $d$-simplex and let $\alpha$ be a multiindex of length $d+1$. Monomials in the barycentric coordinates are compactly written as

$$\mathbf{b}_d^\alpha = \Pi_{i=1}^{d+1} b_i^{\alpha_i}$$

The Bernstein polynomials of degree $n$ on the $d$-simplex are defined by

$$B_\alpha^n = \frac{n!}{\alpha!}\mathbf{b}_d^\alpha, \tag{4}$$

where $|\alpha| = n$. The complete set of Bernstein polynoimals,

$$\{B_\alpha^n\}_{|\alpha|=n},$$

form a basis of polynomials of complete degree $n$ on $S_d$. They form a nonnegative partition of unity on $S_d$.

At one point later, we will also need the *degree elevation* operator [16,14]. If $n > 0$ and $u$ is a polynomial of degree $n-1$, then

$$u = \sum_{|\alpha|=n-1} c_\alpha B_\alpha^{n-1}$$

for some vector of coefficients $c_\alpha$. However, $u$ is also a polynomial of degree $n$ and so may be expressed as

$$u = \sum_{|\alpha|=n} \tilde{c}_\alpha B_\alpha^n$$

for some vector $\tilde{c}_\alpha$. The process of obtaining $\tilde{c}$ from $c$ is called *degree elevation*. Degree elevation is naturally represented as a linear transformation $\tilde{c} = E^{d,n}c$ whose rows correspond to multiindeices with $|\alpha| = n$ and columns to multiindices with $|\beta| = n-1$. From basic linear algebra, the column of $E^{d,n}$ corresponding to some $|\beta| = n-1$ expresses $B_\beta^{n-1}$ as a linear combination of $\{B_\alpha^n\}_{|\alpha|=n}$. It is well-known [16,14] that $E^{d,n}$ contains no more than $d+1$ nonzero entries per column, independent of $n$.

Differentiation may also be represented as a sparse matrix for B-form [14]. If $u = \sum_{|\alpha|=n} u_\alpha B_\alpha^n$ is a polynomial of degree $n$ and $s$ is some direction vector, then $\frac{\partial u}{\partial s} = \sum_{|\alpha|=n-1} \hat{u}_\alpha B_\alpha^{n-1}$ is a polynomial of degree $n-1$ whose coefficients may be computed by a row matrix operation

$$\tilde{u} = Du, \tag{5}$$

where $D$ has at most $d+1$ nonzeros per row.

### *2.3 Warped Gauss quadrature rules*

On $[-1,1]$, we let $\{\zeta_i\}_{i=0}^m$ denote the $m+1$-point Gauss quadrature points (the zeros of the Legendre polynomials) and associated weights $\{w_i\}_{i=0}^m$. Let $\{(\zeta_j,\zeta_i)\}_{i,j=0}^m$ be the tensor product of these points on the biunit square $[-1,1]^2$. As discussed in [12], a quadrature rule that

is accurate on bivariate polynomials of total degree $2m$ is obtained by warping these points from $[-1,1]^2$ to the triangle $\hat{T}$ with vertices $(-1,-1)$, $(1,-1)$, and $(-1,1)$ via the mapping

$$\begin{aligned} \xi_1 &= \frac{(1+\eta_1)(1-\eta_2)}{2} - 1 \\ \xi_2 &= \eta_2, \end{aligned} \tag{6}$$

where $\xi_1, \xi_2$ are the Cartesian coordinates on the triangle and $\eta_1, \eta_2$ on the square. This mapping and its inverse are known to preserve polynomials.

Applying this mapping to the Gauss-point tensor product leads to a quadrature rule with points $Z_{ij} = (\frac{(1+\zeta_j)(1-\zeta_i)}{2} - 1, \zeta_i)$, where $0 \leq i, j \leq m$. The associated weights are $w_{ij} = \frac{w_i w_j (1-\eta_i)}{4}$, where a factor of $\frac{1-\eta_i}{2}$ incorporates the Jacobian of the mapping between the square and triangle.

To evaluate polynomials expressed in the Bernstein basis, we will need to convert $Z_{ij}$ to barycentric coordinates. It is not hard to show that the barycentric coordinates for the triangle $\hat{T}$ are given by

$$(\xi_1, \xi_2) \mapsto \left( \frac{1}{2}(1-\xi_1) - \frac{1}{2}(1+\xi_2), \frac{1}{2}(1+\xi_1), \frac{1}{2}(1+\xi_2) \right),$$

calling the three terms in the triple $(b_1, b_2, b_3)$.

Now, if we apply this mapping to each point $Z_{ij}$, we obtain

$$Z_{ij} = (b_{1ij}, b_{2ij}, b_{3ij}), \tag{7}$$

where

$$
\begin{aligned}
b_{1ij} &= \frac{1}{2}\left(1-\left(\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_i\right)-1\right)\right)-\frac{1}{2}\left(1+\zeta_i\right),\\
b_{2ij} &= \frac{1}{2}\left(1+\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_i\right)-1\right),\\
b_{3ij} &= \frac{1}{2}\left(1+\zeta_i\right).
\end{aligned}
$$

Some simplification is in order.

$$
\begin{aligned}
b_{1ij} &= \frac{1}{2}\left(1-\left(\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_i\right)-1\right)\right)-\frac{1}{2}\left(1+\zeta_i\right)\\
&= \frac{1}{2}\left(2-\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_i\right)\right)-\frac{1}{2}\left(1+\zeta_i\right)\\
&= 1-\frac{1}{4}\left(1+\zeta_j\right)\left(1-\zeta_i\right)-\frac{1}{2}\left(1+\zeta_i\right)\\
&= 1-\frac{1}{4}\left(1-\zeta_i+\zeta_j-\zeta_i\zeta_j\right)-\frac{1}{2}\left(1+\zeta_i\right)\\
&= \frac{1}{4}\left(1-\zeta_i-\zeta_j+\zeta_i\zeta_j\right)\\
&= \frac{1}{4}\left(1-\zeta_i\right)\left(1-\zeta_j\right),
\end{aligned}
$$

and

$$
\begin{aligned}
b_{2ij} &= \frac{1}{2}\left(1+\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_i\right)-1\right)\\
&= \frac{1}{4}\left(1-\zeta_i\right)\left(1+\zeta_j\right),
\end{aligned}
$$

so that

$$
Z_{ij} = \left(\frac{1}{4}\left(1-\zeta_i\right)\left(1-\zeta_j\right), \frac{1}{4}\left(1-\zeta_i\right)\left(1+\zeta_j\right), \frac{1}{2}\left(1+\zeta_i\right)\right). \tag{8}
$$

Note that the third coordinate $b_{3ij}$ only depends on $i$, leaving blocks of quadrature points with the third barycentric coordinate

constant. It will be more convenient to have the first barycentric coordinate constant over blocks. Fortunately, permutating the barycentric coordinates of a quadrature rule leaves the order of accuracy unchanged. We state this with proof for triangles, but it is also true for higher-dimensional simplices.

**Lemma 1** *Let $T$ be a triangle and $(x(b_1,b_2,b_3), y(b_1,b_2,b_3))$ be the mapping from barycentric to Cartesian coordinates. Let $\{w_i,(b_{1i},b_{2i},b_{3i})\}_{i=1}^N$ be the weights and barycentric coordinates of a quadrature rule such that*

$$\int_T p(x)\mathrm{dx} \approx |T| \sum_{i=1}^{N} w_i p\left(x\left(b_{1i},b_{2i},b_{3i}\right), y\left(b_{1i},b_{2i},b_{3i}\right)\right)$$

*is accurate for polynomials of degree $k$. Let $\sigma$ be any permutation of $\{1,2,3\}$. Then the quadrature rule $\left\{w_i,\left(b_{\sigma(1)i},b_{\sigma(2)i},b_{\sigma(3)i}\right)\right\}_{i=1}^N$ is also accurate for all polynomials of degree $k$.*

*Proof* The result is true if and only if it is true on a basis. In particular, we will show the results holds on the Bernstein basis. It is known [16] that for $|\alpha| = k$ and $d = 2, 3$

$$\int_T b^{\alpha_1} b^{\alpha_2} b^{\alpha_3}\, dx = \frac{2|T|\alpha!}{(k+2)!},$$

from which we obtain that the integral of any Bernstein polynomial of degree $k$ is

$$\int_T B_\alpha^k\, dx = \frac{2|T|}{(k+1)(k+2)},$$

which in depends only on $|\alpha| = k$ and not the particular entries of $\alpha$. So, fix any $\alpha$ with $|\alpha| = k$. Then

$$\frac{2|T|}{(k+1)(k+2)} = \int_T B_\alpha^k \, dx = |T| \sum_{i=1}^N w_i \frac{k!}{\alpha!} b_{1i}^{\alpha_1} b_{2i}^{\alpha_2} b_{3i}^{\alpha_3}$$

On the other hand, let $\sigma(\alpha) = \left(\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, \alpha_{\sigma(3)}\right)$ be a permutation of the multiindex $\alpha$, and consider $B_{\sigma(\alpha)}^k$. Then

$$\begin{aligned}
\int_T B_\alpha^k \, dx &= \frac{2|T|}{(k+1)(k+2)} \\
&= \int_T B_{\sigma(\alpha)}^k \, dx \\
&= |T| \sum_{i=1}^N w_i \frac{k!}{\sigma(\alpha)!} b_{1i}^{\alpha_{\sigma(1)}} b_{2i}^{\alpha_{\sigma(2)}} b_{3i}^{\alpha_{\sigma(3)}} \\
&= |T| \sum_{i=1}^N w_i \frac{k!}{\alpha!} b_{\sigma(1)i}^{\alpha_1} b_{\sigma(2)i}^{\alpha_2} b_{\sigma(3)i}^{\alpha_3}.
\end{aligned}$$

With this result in hand, we will take our triangular quadrature rule to be a permutation of (8):

$$Z_{ij} = \left( \frac{1}{2}(1+\zeta_i), \frac{1}{4}(1-\zeta_i)(1+\zeta_j), \frac{1}{4}(1-\zeta_i)(1-\zeta_j) \right). \tag{9}$$

We will also consider the case of tetrahedra. The collapsed Gauss quadrature rules are constructed by mapping from the triunit cube $[-1,1]^3$ to the tetrahedron with vertices $(-1,-1,-1)$, $(1,-1,-1)$, $(-1,1,-1)$, and $(-1,-1,1)$ using

$$\begin{aligned}
\xi_1 &= \frac{1}{4}(1+\eta_1)(1-\eta_2)(1-\eta_3) - 1, \\
\xi_2 &= \frac{1}{2}(1+\eta_2)(1-\eta_3) - 1, \\
\xi_3 &= \eta_3.
\end{aligned} \tag{10}$$

For more discussion of this mapping, see [12].

Warping the tensor product Gauss points $\{(\zeta_k, \zeta_j, \zeta_i)\}_{0\leq i,j,k\leq m}$ with this mapping gives the points

$$Z_{ijk} = \left(\frac{1}{4}(1+\zeta_k)(1-\zeta_j)(1-\zeta_i) - 1, \frac{1}{2}(1+\zeta_j)(1-\zeta_i) - 1, \zeta_i\right) \tag{11}$$

The barycentric coordinates are given by

$$\begin{aligned} b_1 &= -\frac{1}{2}(1+\xi_1+\xi_2+\xi_3), \\ b_2 &= \frac{1}{2}(1+\xi_1), \\ b_3 &= \frac{1}{2}(1+\xi_2), \\ b_4 &= \frac{1}{2}(1+\xi_3). \end{aligned} \tag{12}$$

Now, we compute the barycentric representation of the warped Gauss points to obtain $Z_{ijk} = (b_{1ijk}, b_{2ijk}, b_{3ijk}, b_{4ijk})$ with

$$\begin{aligned} b_{1ijk} &= -\frac{1}{2}\left(1 + \frac{1}{4}(1+\zeta_k)(1-\zeta_j)(1-\zeta_i) - 1 + \frac{1}{2}(1+\zeta_j)(1-\zeta_i) - 1 + \zeta_i\right) \\ &= -\frac{1}{2}\left(\frac{1}{4}(1+\zeta_k)(1-\zeta_j)(1-\zeta_i) + \frac{1}{2}(1+\zeta_j)(1-\zeta_i) - (1-\zeta_i)\right) \\ &= -\frac{1}{2}(1-\zeta_i)\left(\frac{1}{4}(1+\zeta_k)(1-\zeta_j) + \frac{1}{2}(1+\zeta_j) - 1\right) \\ &= \frac{1}{8}(1-\zeta_i)(1-\zeta_j)(1-\zeta_k), \end{aligned} \tag{13}$$

where we have simplified the last set of parentheses on the next to the last line by

$$\begin{aligned}
&\frac{1}{4}\left(1+\zeta_k\right)\left(1-\zeta_j\right)+\frac{1}{2}\left(1+\zeta_j\right)-1\\
=&\frac{1}{4}\left(1-\zeta_j+\zeta_k-\zeta_j\zeta_k\right)+\frac{1}{2}+\frac{1}{2}\zeta_j-1\\
=&-\frac{1}{4}+\frac{1}{4}\zeta_j+\frac{1}{4}\zeta_k-\frac{1}{4}\zeta_j\zeta_k\\
=&-\frac{1}{4}\left(1-\zeta_j\right)\left(1-\zeta_k\right).
\end{aligned}$$

We also have

$$\begin{aligned}
b_{2ijk}&=\frac{1}{2}\left(1+\frac{1}{4}\left(1+\zeta_k\right)\left(1-\zeta_j\right)\left(1-\zeta_i\right)-1\right)\\
&=\frac{1}{8}\left(1+\zeta_k\right)\left(1-\zeta_j\right)\left(1-\zeta_i\right),
\end{aligned}\tag{14}$$

$$\begin{aligned}
b_{3ijk}&=\frac{1}{2}\left(1+\frac{1}{2}\left(1+\zeta_j\right)\left(1-\zeta_1\right)-1\right)\\
&=\frac{1}{4}\left(1+\zeta_j\right)\left(1-\zeta_i\right),
\end{aligned}\tag{15}$$

and

$$b_{4ijk}=\frac{1}{2}\left(1+\zeta_i\right).\tag{16}$$

An analog of Lemma 1 holds for tetrahedra, so we will permute the barycentric coordinates to

$$\begin{aligned}
Z_{ijk}&=\left(b_{1ijk},b_{2ijk},b_{3ijk},b_{4ijk}\right)\\
&=\left(\frac{1}{2}\left(1+\zeta_i\right),\frac{1}{4}\left(1+\zeta_j\right)\left(1-\zeta_i\right),\frac{1}{8}\left(1+\zeta_k\right)\left(1-\zeta_j\right)\left(1-\zeta_i\right),\right.\\
&\quad\left.\frac{1}{8}\left(1-\zeta_k\right)\left(1-\zeta_j\right)\left(1-\zeta_i\right)\right).
\end{aligned}\tag{17}$$

## 3 Bernstein-Vandermonde-Gauss matrices

**Definition 1** *Let $1 \leq d \leq 3$, $m, n \geq 0$ be integers, and $Z^m$ the set of Gauss points ($d = 1$) or warped Gauss points ($d = 2, 3$). The* Bernstein-Vandermonde-Gauss matrix *is defined as*

$$V_{I,\alpha}^{d,m,n} = B_\alpha^n(Z_I), \tag{18}$$

*where $|\alpha| = n$ and all components of $I$ are no greater than $m$.*

For the univariate case, Bernstein-Vandermonde matrices have been studied in the literature. Despite possibly large condition numbers, stable and efficient algorithms for basic linear algebra problems are known [15, 1]. Our goal is to investigate substructure in the higher-dimensional matrices that enables them to be multiplied onto vectors efficiently.

Before giving explicit formulas for these higher-dimensional matrices, we describe the one-dimensional matrix tabulated at the Gauss points. As before, let $\{\zeta_i\}$ be the Gauss points on $[-1, 1]$. If these are converted to barycentric coordinates , we obtain

$$\left\{ \left( \frac{1-\zeta_i}{2}, \frac{1+\zeta_i}{2} \right) \right\}_{i=0}^{m}$$

For the one-dimensional Bernstein-Vandermonde-Gauss matrix we will use index rather than multiindex notation for the polynomials.

The matrix is

$$V_{i,j}^{1,m,n} = \frac{n!}{j!(n-j)!} \left(\frac{1-\zeta_i}{2}\right)^{n-j} \left(\frac{1+\zeta_i}{2}\right)^{j}, \tag{19}$$

We will assume hereafter that $V^{1,m,n}$ and its transpose may be stably applied in $\mathcal{O}(mn)$ operations.

Now, we consider the two- and three-dimensional cases. By using (4) and (9), the two-dimensional Bernstein-Vandermonde-Gauss is

$$\begin{aligned} V_{I,\alpha}^{2,m,n} &= B_\alpha^n(Z_I) \\ &= \frac{n!}{\alpha!} \left(\frac{1}{2}(1+\zeta_i)\right)^{\alpha_1} \left(\frac{1}{4}(1-\zeta_i)(1+\zeta_j)\right)^{\alpha_2} \left(\frac{1}{4}(1-\zeta_i)(1-\zeta_j)\right)^{\alpha_3}, \end{aligned} \tag{20}$$

and (4) and (11) together give the three-dimensional matrix as

$$\begin{aligned} V_{I,\alpha}^{3,m,n} &= B_\alpha^n(Z_I) \\ &= \frac{n!}{\alpha!} \left(\frac{1}{2}(1+\zeta_i)\right)^{\alpha_1} \left(\frac{1}{4}(1-\zeta_i)(1+\zeta_j)\right)^{\alpha_2} \\ &\quad \times \left(\frac{1}{8}(1-\zeta_i)(1-\zeta_j)(1+\zeta_k)\right)^{\alpha_3} \left(\frac{1}{8}(1-\zeta_i)(1-\zeta_j)(1-\zeta_k)\right)^{\alpha_4} \end{aligned} \tag{21}$$

These matrices posses some important structure that we explore before presenting fast algorithms in the following two sections.

Consider the equation for $V_{I,\alpha}^{2,m,n}$. Since $n - \alpha_1 = \alpha_2 + \alpha_3$, we may factor all the terms with $1 \pm \zeta_i$ to write

$$V_{I,\alpha}^{2,m,n} = \frac{n!}{\alpha!} \left(\frac{1}{2}(1+\zeta_i)\right)^{\alpha_1} \left(\frac{1}{2}(1-\zeta_i)\right)^{n-\alpha_1} \left(\frac{1}{2}(1+\zeta_j)\right)^{\alpha_2} \left(\frac{1}{2}(1-\zeta_j)\right)^{\alpha_3}.$$

Now, if we multiply and divide by $(n-\alpha_1)!$, we may regroup the factors

$$V_{I,\alpha}^{2,m,n} = \frac{n!}{\alpha_1!\,(n-\alpha_1)!}\left(\frac{1}{2}\,(1+\zeta_i)\right)^{\alpha_1}\left(\frac{1}{2}\,(1-\zeta_i)\right)^{n-\alpha_1} \times \frac{(n-\alpha_1)!}{\alpha_2!\alpha_3!}\left(\frac{1}{2}\,(1+\zeta_j)\right)^{\alpha_2}\left(\frac{1}{2}\,(1-\zeta_j)\right)^{\alpha_3}.$$

Now, we use that $\alpha_3 = n-\alpha_1-\alpha_2$ and rewrite factors as binomial coefficients to obtain

$$\begin{aligned} V_{I,\alpha}^{2,m,n} = &\binom{n}{\alpha_1}\left(\frac{1}{2}\,(1+\zeta_i)\right)^{\alpha_1}\left(\frac{1}{2}\,(1-\zeta_i)\right)^{n-\alpha_1} \\ &\times \binom{n-\alpha_1}{\alpha_2}\left(\frac{1}{2}\,(1+\zeta_j)\right)^{\alpha_2}\left(\frac{1}{2}\,(1-\zeta_j)\right)^{n-\alpha_1-\alpha_2}, \end{aligned} \tag{22}$$

which becomes

$$V_{I,\alpha}^{2,m,n} = V_{i,\alpha_1}^{1,m,n} V_{j,\alpha_2}^{1,m,n-\alpha_1} \tag{23}$$

by using (19).

To interpret this calculation, we may think of partitioning $V^{2,m,n}$ into blocks corresponding to fixing $i$ and $\alpha_1$. Each such block consists of $m+1$ rows and $n-\alpha_1$ columns, and is a scaled version of a one-dimensional operator. If we denote these blocks as $V_{i,\alpha_1}^{2,m,n} \in \mathbb{R}^{m+1,n-\alpha_1}$, we may write

$$V_{i,\alpha_1}^{2,m,n} = V_{i,\alpha_1}^{1,m,n} V^{1,m,n-\alpha_1}. \tag{24}$$

Now, a similar calculation may be carried out for the tetrahedral Bernstein-Vandermonde-Gauss matrix. First we factor terms with

$(1 \pm \zeta_i)$ and multiply by $\frac{(n-\alpha_1)!}{(n-\alpha_1)!}$ to obtain

$$\begin{aligned} V_{I,\alpha}^{3,m,n} &= \frac{n!}{\alpha!}\left(\frac{1}{2}(1+\zeta_i)\right)^{\alpha_1}\left(\frac{1}{4}(1-\zeta_i)(1+\zeta_j)\right)^{\alpha_2} \\ &\quad\times\left(\frac{1}{8}(1-\zeta_i)(1-\zeta_j)(1+\zeta_k)\right)^{\alpha_3}\left(\frac{1}{8}(1-\zeta_i)(1-\zeta_j)(1-\zeta_k)\right)^{\alpha_4} \\ &= \frac{n!}{\alpha_1!(n-\alpha_1)!}\left(\frac{1}{2}(1+\zeta_i)\right)^{\alpha_1}\left(\frac{1}{2}(1-\zeta_i)\right)^{n-\alpha_1} \\ &\quad\times\frac{(n-\alpha_1)!}{\alpha_2!\alpha_3!\alpha_4!}\left(\frac{1}{2}(1+\zeta_j)\right)^{\alpha_2}\left(\frac{1}{4}(1-\zeta_j)(1+\zeta_k)\right)^{\alpha_3} \\ &\quad\times\left(\frac{1}{4}(1-\zeta_j)(1-\zeta_k)\right)^{\alpha_4}. \end{aligned} \tag{25}$$

We could continue the process by grouping terms with $(1 \pm \zeta_j)$, but instead we recognize using (19) and (20) that

$$V_{I,\alpha}^{3,m,n} = V_{i,\alpha_1}^{1,m,n} V_{\tilde{I},\tilde{\alpha}}^{2,m,n-\alpha_1}, \tag{26}$$

where $\tilde{I} = (j,k)$ and $\tilde{\alpha} = (\alpha_2, \alpha_3, \alpha_4)$ with $|\tilde{\alpha}| = n - \alpha$.

As with $V^{2,m,n}$, we may view $V^{3,m,n}$ as a blocked matrix, with blocks obtained by fixing the first indices $i$ and $\alpha_1$. For each $i$ and $\alpha_1$, we write $V_{i,\alpha_1}^{3,m,n} \in \mathbb{R}^{(m+1)^2,\binom{n-\alpha_1+1}{2}}$ as

$$V_{i,\alpha_1}^{3,m,n} = V_{i,\alpha_1}^{1,m,n} V^{2,m,n-\alpha_1} \tag{27}$$

The calculations leading to (24) and (27), then, have shown that

**Theorem 1** *Let $1 < d \leq 3$ and $m, n \geq 0$. Let $I = (i_1, i_2 \ldots, i_d)$ with $0 \leq i_1, i_2 \ldots, i_d \leq m$ and let $|\alpha| = n$. Let $\tilde{I} = (i_2, \ldots, i_d)$ and*

$\tilde{\alpha} = (\alpha_2, \ldots, \alpha_d)$. *The Bernstein-Vandermonde-Gauss matrices are naturally blocked according to*

$$V^{d,m,n}_{I,\alpha} = V^{1,m,n}_{i,\alpha_1} V^{d-1,m,n-\alpha_1}_{\tilde{I},\tilde{\alpha}}. \tag{28}$$

Theorem 1 shows that the higher-dimensional Bernstein-Vandermonde-Gauss matrices have a natural structure similar to a Kronecker product. A standard Kronecker product $A \otimes B$ constructs a block matrix with scaled copies of $B$ tiled in each block, so all the blocks are the same size. Our case has variable block size, and a slightly different matrix in each block of columns. Before studying this in further detail, we give another property of the BVG matrices.

**Proposition 1** *Let $d \geq 1$ and $m \geq 0$ and $n > 0$ and let $E^{d,n}$ denote the Bernstein degree elevation operator from polynomials of degree $n-1$ to degree $n$. Then the Vandermonde matrices satisfy*

$$V^{d,m,n-1} = V^{d,m,n} E^{d,n}. \tag{29}$$

*Proof* Since $V^{d,m,n-1}$ evaluates any B-form polynomial of degree $n-1$ and $V^{d,m,n}$ evaluates any B-form polynomial of degree $n$ at the same points, this result is obvious: if one elevates the degree (while preserving the polynomial) and evaluates it at the same points, one obtains the same values.

## 4 Discrete function tabulation

Let $u = \sum_{|\alpha|=n} u_\alpha B_\alpha^n$ be a function expressed in the Bernstein basis. Consider the problem of evaluating $u$ at all of the warped Gauss points. This is accomplished by forming the product $V^{d,m,n}u$, where $V$ is given above in (19). The standard algorithm requires work proportial to the product of the number of quadrature points and the dimension of the polynomial space. Efficient algorithms, however, follow immediately from Theorem 1. A blocked version of the standard matrix-vector algorithm could be

$$y_i = \sum_{\alpha_1=0}^{n} V_{i,\alpha_1}^{d,m,n} u_{\alpha_1}. \tag{30}$$

Here, $u_{\alpha_1}$ is the subvector of $u$ consisting of all entries with first multiindex fixed as $\alpha_1$, and similar for $y_i$. Theorem 1 allows us to replace the matrix block $V_{i,\alpha_1}^{d,m,n}$ with a scaled copy of $V^{d-1,m,n-\alpha_1}$. Since each such $V_{i,\alpha_1}^{d,m,n}$ block acts on $u_{\alpha_1}$, we may compute one matrix-vector product

$$V^{d-1,m,n-\alpha_1} u_{\alpha_1}$$

and add a scaled version of it to each of the $m+1$ chunks of $y$. This gives Algorithm 2.

If $d = 2$, then Algorithm 2 requires $n+1$ applications of one-dimensional Bernstein Vandermonde matrices. If $m \approx n$, then this

**Algorithm 2** Applies the Bernstein-Vandermonde-Gauss matrix $V^{d,m,n}$ with $d = 2, 3$ to a vector $u$

$y \leftarrow 0$

**for** $\alpha_1 \leftarrow 0,n$ **do**

  $x \leftarrow V^{d-1,m,n-\alpha_1} u_{\alpha_1}$

  **for** $i \leftarrow 0,m$ **do**

    $y_i \leftarrow y_i + V^{1,m,n}_{i,\alpha_1} x$

  **end for**

**end for**

gives $\mathcal{O}(n^3)$ operations. Additionally, the inner loop (scaling and addition) is executed $\mathcal{O}(n^2)$ times, each with a cost of $\mathcal{O}(n)$. This also gives $\mathcal{O}(n^3)$ operations. Now, consider three dimensions. In this case, we have $n+1$ applications of the two-dimensional matrix, each costing $\mathcal{O}(n^3)$ operations for $\mathcal{O}(n^4)$ operations. The inner loop is again executed $\mathcal{O}(n^2)$ times, and each time now costs $\mathcal{O}(n^2)$ operations. To summarize,

**Theorem 2** *For $m = \mathcal{O}(n)$ and $d = 1, 2, 3$, Algorithm 2 requires $\mathcal{O}(n^{d+1})$ floating-point operations to execute.*

## 5 Finite element integration

So far, we have seen how to tabulate finite element functions at a particular set of quadrature points efficiently. This is the first step

in evaluating the action of bilinear forms on a single element. Now, suppose we need to compute

$$\int_T f(x) B_\alpha^n \, dx \tag{31}$$

for all $|\alpha| = n$ and that $f$ is already known at the warped Gauss points on a simplex $T$, whether by direct evaluation of a given function or use of the techniques in the previous section. Let $f_I$ be the vector of values of $f$ tabulated at the quadrature points, and let $\tilde{f}_I$ be the values of $f$ times the quadrature weights at each point. Alternatively, if $w_I$ is the quadrature weight associated with a point, then $\tilde{f} = Wf$ for a diagonal matrix with $w_I$ on the diagonal.

Evaluating (31) for all Bernstein polynomials is obtained by the matrix-vector product.

$$\left(V^{d,m,n}\right)^t \tilde{f}. \tag{32}$$

A simple blocked algorithm for the product $y = \left(V^{d,m,n}\right)^t u$ starts by writing the product as

$$y_{\alpha_i} = \sum_{i=0}^{m} \left(V_{i,\alpha_1}^{d,m,n}\right)^t u_i.$$

As before, Theorem 1 lets us write the term inside the sum as

$$V_{i,\alpha_1}^{1,m,n-\alpha_1} \left(V^{d-1,m,n-\alpha_1}\right)^t u_i.$$

After using Theorem 1 in applying $V^{d,m,n}$, each block of $u$ was multiplied by the block $V^{d-1,m,n-\alpha_1}$ several times. Now, the transposes

of these lower-dimensional matrices act on all the blocks of the input vector $u$. A fast algorithm, then, is more subtle than the simple loop-hoisting technique used in Algorithm 2. Instead, much as in [14], we will use the sparsity of the degree elevation operator to obtain an efficient algorithm.

For $\alpha_1 = 0$, we can use Proposition 1 to rewrite the product in the sum as

$$V_{i,0}^{1,m,n-1} \left(E^{d-1,n}\right)^t \left(V^{d-1,m,n}\right)^t u_i.$$

Hence, after computing

$$\left(V^{d-1,m,n}\right)^t u_i,$$

we may successively apply the transposed degree elevation operators to its result, scaling the result. Consequently, we may compute

$$\left(V^{d-1,m,n-\alpha_1-1}\right)^t u_i$$

recursively by

$$\left(E^{d-1,n-\alpha_1}\right)^t \left(V^{d-1,m,n-\alpha_1}\right)^t u_i$$

This gives rise to Algorithm 3.

Degree elevation has linear complexity, so a similar discussion as for Algorithm 2 gives the complexity result

**Theorem 3** *For $m = \mathcal{O}(n)$ and $d = 1, 2, 3$, Algorithm 3 requires $\mathcal{O}(n^{d+1})$ floating-point operations to execute.*

**Algorithm 3** Computes $y = \left(V^{d,m,n}\right)^t u$, where $d \geq 2$.

$y \leftarrow 0$

**for** $i \leftarrow 0,m$ **do**

$x \leftarrow \left(V^{d-1,m,n-\alpha_1}\right)^t u_i$

**for** $\alpha_1 \leftarrow 0,n$ **do**

$y_{\alpha_1} \leftarrow y_{\alpha_1} + V^{1,m,n}_{i,\alpha_1} x$

$x \leftarrow \left(E^{d-1,n-\alpha_1}\right)^t x$

**end for**

**end for**

## 6 Bilinear form evaluation

We now return to the question of evaluating the action of bilinear forms (1) by our fast algorithms for applying $V^{d,m,n}$ and its transpose.

Let $u$ be a polynomial of degree $n_1$ and suppose we wish to calculate

$$(Ku)_\alpha = \int_T \kappa(x) \left(D_1 u\right) \left(D_2 B^{n_2}_\alpha\right) \, dx \tag{33}$$

for all $|\alpha| = n_2$. We will use $u$ to denote both the polynomial and its vector of coefficients, as indicated in context.

We will evaluate this by the warped Gauss numerical quadrature of order $m$, where $m$ is sufficiently large to ensure the right convergence rate in the finite element method [7].

First, we compute the B-form coefficients of $y_1 = D_1 u$ by the sparse matrix-vector product (5). This gives a polynomial of degree

$n_1 - 1$, which we must evaluate at the the Gauss-quadrature points. This is accomplished by the operation

$$V^{d,m,n_1-1} y_2.$$

Next, this result is multiplied pointwise by the product of the quadrature weights and the coefficient function $\kappa$ evaluated at each quadrature point. This is represented by a matrix product

$$y_3 = \tilde{W} y_2.$$

As described above, the product

$$y_4 = \left(V^{d,m,n_2}\right)^t y_3$$

now contains the integrals of $D_1 u$ against all of the degree $n_2$ Bernstein basis functions, and the calculation is completed by

$$y_5 = (D_2)^t y_4,$$

which incorporates the differential operator $D_2$. The whole calculation is written by

$$Ku = (D_2)^t \left(V^{d,m,n_2}\right)^t \tilde{W} V^{d,m,n_1} D_1 u.$$

Both $D_1$ and $D_2$ are sparse, with $d+1$ nonzeros per row, and $\tilde{W}$ is diagonal. The dominant cost in this product is applying $V$ and $V^t$, and we have presented low-complexity algorithms for these.

## 7 Numerical results

Here, we will test the performance of our algorithms on a few sample problems. We want to evaluate the cost to apply the assembled matrix versus the matrix-free approach. We use pieces of the Sundance package [17,13] within Trilinos, such as the mesh, degree of freedom mappings, and interface to Epetra vector operations [10], but not its symbolic interface.

To demonstrate the scaling of our operations, we considered both the constant and variable coefficient Poisson operators, arising from the discretization of the variational forms

$$\int_\Omega \nabla u \cdot \nabla v \, dx$$

and

$$\int_\Omega w \nabla u \cdot \nabla v \, dx,$$

respectively. In the latter case, the function $w$ is chosen as the $L^2$ projection of some existing function into the finite element space.

In our numerical experiments, we meshed the unit square into a $64 \times 64$ grid of squares, each subdivided into two right triangles. We used Sundance's degree of freedom mappings to implement the scatter/gather operation $\mathcal{A}$. We have implemented C++ code that performs the differentiation operations and the Vandermonde matrix application and its transpose. On each element, the constant

coefficient Poisson operator requires two derivative operations, two transposed derivative operations, two Bernstein-Vandermonde matrix applications, and two transposed applications. The variable coefficient operator requires an additional application of a Bernstein-Vandermonde matrix to evaluate the coefficient function $w$, at the quadrature points. All our timings, reported in Table 1, were obtained on a single core of MacPro with dual 2.8 GHz Xeon quad-core processors and 32GB of RAM.

We compared the matrix-free methods to constructing local stiffness matrices for all the cells and assembling into two matrix formats. For one, we modeled a sparse matrix by a `vector<map<unsigned,double> >` from C++ the standard template library. In this case, each row is stored as an associated array mapping nonzero column entries to the double-precision values. To compute the matrix-vector product, we used an iterator to traverse the nonzero entries of each row. As a second more optimized case, we used the`Epetra_FECrsMatrix` from the Epetra package within Trilinos [10]. This is a specialization of the standard sparse matrix implementation to support direct summation of element matrices into the sparse data stucture. The third column Table 1 shows the time required to compute all of the element stiffness matrices using the In-

| | | | STL | | Epetra | |
|---|---|---|---|---|---|---|
| $k$ | Bernstein | Cell mat | Assembly | Matvec | Assembly | Matvec |
| 1 | 0.0024339 | 0.0073817 | 0.0003429 | 0.0078258 | 2.40E-005 | 0.002937 |
| 2 | 0.0068065 | 0.0220394 | 0.0055234 | 0.0561759 | 4.90E-005 | 0.012849 |
| 3 | 0.0132358 | 0.0693899 | 0.0184985 | 0.184349 | 9.20E-005 | 0.033307 |
| 4 | 0.0257605 | 0.230018 | 0.046973 | 0.462899 | 0.000152 | 0.073813 |
| 5 | 0.0399169 | 0.466646 | 0.0974356 | 0.949906 | 0.000225 | 0.139959 |
| 6 | 0.0600805 | 0.864955 | 0.185518 | 1.7523 | 0.000321 | 0.230152 |
| 7 | 0.0901709 | 1.56259 | 0.306342 | 2.99176 | 0.000459 | 0.39826 |
| 8 | 0.122464 | 2.62408 | 0.504124 | 4.87428 | 0.000631 | 0.646607 |
| 9 | 0.163513 | 4.27394 | 0.738311 | 7.23924 | 0.00092 | 0.899606 |
| 10 | 0.212413 | 4.99079 | 1.05475 | 11.7651 | 0.001438 | 1.24465 |

**Table 1.** For various polynomial degree (first column), time to perform matrix-free application of the Poisson operator using Algorithms 2 and 3 plus a scatter and gather operation (second column). These times are compared to the cost of constructing all element stiffness matrices (third column), assembling them into either STL and Epetra format (columns four and six), and computing the matrix-vector product (columns three and five).

trepid package within Trilinos [6], and the final columns display the time to assemble these element matrices into the sparse data structure and compute the matrix-vector product for our two sparse data structures.

An examination of the relative times in Table 1 reveals several interesting features. First, in all cases, the cost of a matrix-vector product is smaller than the cost of building element stiffness matrices and assembling them into a sparse data structure, and the difference becomes quite dramatic as the order increases. While the matrix-free method surpasses the STL-based sparse matrix at cubics, the Epetra matrix-vector product is highly optimized, and we are yet to be competitive with this. However, the savings in bypassing the matrix construction and the massive reduction in memory footprint are still significant advantages to our method over either sparse matrix approach. We hope to better tune our Bernstein implementation in future work.

We also tested our algorithm for a weighted Poisson problem in which the coefficient lived in the finite element space. Not much changes in this situation – our matrix-free algorithm is about 50% more expensive in this case, but still surpasses the STL-based matrix-vector product for STL-based by cubics. The element construction is slightly more expensive, and the matrix-vector and assembly costs are the same for the STL and Epetra matrices.

## 8 Conclusions

We have extended our previous low-complexity techniques for constant coefficient operators to quadrature-based forms with possibly variable coefficient by means of finding block structure in simplicial Bernstein-Vandermonde-Gauss matrices, providing a reference implementation that delivers sub-quadratic complexity. These techniques will be applied and extended in several ways in ongoing work. For one, the exterior calculus bases in [3] that generalize Raviart-Thomas and Nedelec elements are expressed in terms of Bernstein-type polynomials, so converting the vector components of these bases to B-form should open up opportunities for spectral-type methods for $H(\mathrm{div})$ and $H(\mathrm{curl})$. Moreover, techniques for variable coefficients can often be extended to work for curvilinear geometry. Additionally, we need to find ways to tune our implementation of the application of $V$ and $V^t$.

## References

1. Jose-Javier Martinex Ana Marco, *Accurate numerical linear algebra with Bernstein-Vandermonde matrices.* arXiv:0812.3115v1, December 2008.
2. Douglas N. Arnold, Richard S. Falk, and Ragnar Winther, *Finite element exterior calculus, homological techniques, and applications*, Acta Numer., 15 (2006), pp. 1–155.

3. ——, *Geometric decompositions and local bases for spaces of finite element differential forms*, Comput. Methods Appl. Mech. Engrg., 198 (2009), pp. 1660–1672.

4. G. Awanou and M.J. Lai, *Trivariate spline approximations of 3d navier-stokes equations*, Mathematics of computation, 74 (2005), pp. 585–602.

5. G. Awanou, M. J. Lai, and P. Wenston, *The multivariate spline method for numerical solution of partial differential equations and scattered data interpolation*, in Wavelets and Splines: Athens 2005, G. Chen and M. J. Lai, eds., Nashboro Press, 2006, pp. 24–74.

6. Pavel Bochev, Denis Ridzal, and Kara Peterson, *Intrepid: Interoperable Tools for Compatible Discretizations*. `http://trilinos.sandia.gov/packages/intrepid`.

7. Susanne C. Brenner and L. Ridgway Scott, *The mathematical theory of finite element methods*, vol. 15 of Texts in Applied Mathematics, Springer, New York, third ed., 2008.

8. Claudio Canuto, M. Yousuff Hussaini, Alfio Quarteroni, and Thomas A. Zang, *Spectral methods in fluid dynamics*, Springer Series in Computational Physics, Springer-Verlag, New York, 1988.

9. Daniele Funaro, *Spectral elements for transport-dominated equations*, vol. 1 of Lecture Notes in Computational Science and Engineering, Springer-Verlag, Berlin, 1997.

10. Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willen-

bring, Alan Williams, and Kendall S. Stanley, *An overview of the trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

11. X.L. Hu, D.F. Han, and M.J. Lai, *Bivariate splines of various degrees for numerical solution of partial differential equations*, SIAM Journal on Scientific Computing, 29 (2008), p. 1338.
12. George Em Karniadakis and Spencer J. Sherwin, *Spectral/hp element methods for computational fluid dynamics*, Numerical Mathematics and Scientific Computation, Oxford University Press, New York, second ed., 2005.
13. Robert C. Kirby Kevin Long and Bart van Bloemen Waanders, *Unified Embedded Parallel Finite Element Computations Via Software-Based Fréchet Differentiation*. submitted to SIAM J. Scientific Computing.
14. Robert C. Kirby, *Fast application of some finite element bilinear forms using bernstein polynomials*, Numerische Mathematik, (to appear).
15. Plamen Koev, *Accurate computations with totally nonnegative matrices*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 731–751 (electronic).
16. Ming-Jun Lai and Larry L. Schumaker, *Spline functions on triangulations*, vol. 110 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, 2007.
17. Kevin Long, *Chapter contribution in "Large-Scale PDE-Constrained Optimization, L. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders editors"*, vol. 30 of Lecture Notes in Computational Science and Engineering, Springer-Verlag, 2003.
18. Larry L. Schumaker, *Computing bivariate splines in scattered data fitting and the finite-element method*, Numer. Algorithms, 48 (2008), pp. 237–260.