SIAM J. SCI. COMPUT.
Vol. 36, No. 2, pp. A846–A868

LOW-COMPLEXITY FINITE ELEMENT ALGORITHMS FOR THE DE RHAM COMPLEX ON SIMPLICES*

ROBERT C. KIRBY†

Abstract. We combine recently developed finite element algorithms based on Bernstein polynomials [M. Ainsworth, G. Andriamaro, and O. Davydov, *SIAM J. Sci. Comput.*, 33 (2011), pp. 3087–3109; R. C. Kirby, *Numer. Math.*, 117 (2011), pp. 631–652] with the explicit basis construction of the finite element exterior calculus [D. N. Arnold, R. S. Falk, and R. Winther, *Comput. Methods Appl. Mech. Engrg.*, 198 (2009), pp. 1660–1672] to give a family of algorithms for the de Rham complex on simplices that achieves stiffness matrix construction and matrix-free action in optimal complexity. These algorithms are based on realizing the exterior calculus bases as short combinations of Bernstein polynomials. Numerical results confirm optimal scaling of the algorithms and favorable comparison with FEniCS at high polynomial order as well. Additional empirical studies show that very high accuracy is achieved in the mixed discretization of the Poisson equation and the Maxwell eigenvalue problem as the polynomial degree increases.

Key words. Bernstein polynomials, spectral element method, finite element exterior calculus

AMS subject classification. 65N30

DOI. 10.1137/130927693

1. Introduction.

Realizing the mathematical power of high-order discretizations of partial differential equations requires specialized algorithms. In rectangular geometry, sum-factorization algorithms dating back to Orszag [28] have proved fundamental, giving optimal-complexity and high arithmetic intensity algorithms for evaluating operators. Though more complicated than in rectangular geometry, the collapsed-coordinate bases discussed extensively in Karniadakis and Sherwin [13] have proved effective in simplicial and other geometries.

Recently, Bernstein polynomials have been proposed as an alternative tool for developing efficient high-order finite element algorithms. Our paper [15] demonstrated that elementwise constant-coefficient mass matrices possess dimensionally recursive blockwise decompositions that lead to optimal-complexity $\mathcal{O}(r^{n+1})$ algorithms for matrix-vector products, where r is the polynomial degree and n the spatial dimension. Because of the sparsity of differentiation in the Bernstein basis, this technique carries over to stiffness matrices. We developed similar techniques applicable to variable coefficients via numerical quadrature in [16]. Around the same time, Ainsworth, Andriamaro, and Davydov also developed techniques for Bernstein polynomials [1]. These are based on the observation that the Duffy transform [10] tensorializes the Bernstein basis, and the resulting algorithms are perhaps more concise to explain and implement than the matrix decompositions in [15, 16]. In addition to optimal complexity for the matrix-vector product, they also give the first known optimal-order $\mathcal{O}(r^{2n})$ algorithm for forming the entries of the element stiffness matrices—constant work per matrix element.

Collapsed-coordinate bases and Bernstein polynomials, then, give two approaches for efficiently implementing standard H^1-type finite element spaces on simplices.

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 5, 2013; accepted for publication (in revised form) February 3, 2014; published electronically April 29, 2014. This work was supported by NSF grant CCF-1117794.
http://www.siam.org/journals/sisc/36-2/92769.html
†Department of Mathematics, Baylor University, Waco, TX 76798-7328 (Robert_Kirby@baylor.edu).

The Sobolev spaces $H(\mathrm{div})$ and $H(\mathrm{curl})$ present additional challenges. While we can refer to the work of Hientzsch [12] in rectangular geometry, we are aware of only [2,3] for simplicial geometry, where Ainsworth, Andriamaro, and Davydov provide a Bernstein-type implementation of the triangular Raviart–Thomas element [30] and their ongoing work to extend this to three dimensions.

In this paper, we take a broader view than [2,3], uniformly handling the de Rham complex. In particular, we take the explicit basis construction [6] provided by Arnold, Falk, and Winther for the de Rham complex [5]. These bases, inspired by earlier work of Gopalakrishnan, García-Castillo, and Demkowicz [11], are based on Bernstein polynomials times Whitney forms. We express all of these bases as geometrically dependent linear combinations of Bernstein polynomials and utilize the already developed fast evaluation and integration techniques. In this way, we are able to give the first family of optimal-complexity algorithms for matrix formation and application that works across the de Rham complex. Linear solvers for higher-order mixed methods in unstructured geometry remain a challenge, however. We point to the work on multigrid for $H(\mathrm{div})$ and $H(\mathrm{curl})$ [4], block preconditioning work for mixed Poisson [29], and the preconditioning techniques for high-order H^1 discretizations in [9] as possible starting points for future work in this direction.

This paper sits in the wider context of our ongoing work to enable effective implementations of finite element methods (FEM). In the FIAT project [14], we offered a computational paradigm based on dense linear algebra that may have been the first realization of arbitrary-order mixed finite element spaces. However general these techniques are, they do not enable optimal-complexity algorithms except for constant coefficient problems, where precomputation of reference integrals is performed. This limitation propagates upward to our work in `ffc` [31]. Likewise, our work in FErari [17] only reduced the number of instructions, typically by some small factor, and did not reduce the order of complexity. This work, though not yet available in FEniCS or other high-level tools, suggests the possibility of obtaining the algorithmic efficiency of spectral element methods while retaining a wide range of possible discretizations in unstructured geometry.

For the rest of this paper, in section 2, we recall notation for Bernstein polynomials on the simplex and recall the fundamental problems of evaluation and moment calculation addressed in earlier papers [1,16]. We recall the complexity results of the algorithms given but refer the reader to those papers for the algorithmic details. We finish section 2 by discussing how to evaluate bilinear forms over linear combinations of Bernstein polynomials. If the linear combinations are short, then the same order of complexity holds for these algorithms. In section 3, we turn to the finite element exterior calculus bases [6] and describe the relevant notation and bases. The major result of this section and the overall paper is that these basis functions and their derivatives are in fact short linear combinations of Bernstein polynomials. In fact, the $P_r^-\Lambda^k$ basis functions consist of $k+1$ Bernstein polynomials, independent of the polynomial degree r and spatial dimension d. Hence, the matrices associated with bilinear forms over these spaces may be assembled and multiplied onto vectors with optimal complexity.

We present some numerical results indicating the performance and accuracy of these bases in section 4. In particular, we give single-core scaling results for matrix assembly and matrix-vector products for both 1- and 2-forms. Also, because Bernstein polynomials can give rise to matrices with very large condition numbers, the accuracy of our methods at high order is a reasonable concern. To this end, we present r-convergence studies on very coarse meshes for the mixed Poisson problem in three dimensions and the curl-curl eigenvalue problem in two and three dimensions.

2. Bernstein-basis finite element algorithms.

2.1. Notation for Bernstein polynomials. Bernstein polynomials are naturally formulated on the n-simplex in terms of the barycentric coordinates and multi-index notation. For a nondegenerate simplex $T \subset \mathbb{R}^n$ with vertices $\{x_i\}_{i=0}^n$, let $\{b_i\}_{i=0}^n$ denote the barycentric coordinates. Each of these is an affine map from $\mathbb{R}^n$ into $\mathbb{R}$ with $b_i(x_j) = \delta_{ij}$ for $0 \le i, j \le n$ and $b_i(x) \ge 0$ for all $x \in T$.

We will also use standard multi-index notation, using lowercase Greek letters to denote multi-indices and beginning the indexing with 0. So, $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_n)$ is a tuple of $n+1$ nonnegative integers. The *order* of α is given by $\sum_{i=0}^n \alpha_i$, and we define the factorial $\alpha! \equiv \prod_{i=0}^n \alpha_i!$. A binomial coefficient with multi-index arguments implies the product over the entries. That is, if $\alpha_i \ge \beta_i$ for each entry i,

$$\binom{\alpha}{\beta} = \prod_{i=0}^n \binom{\alpha_i}{\beta_i}.$$

We also define e_i to be the multi-index consisting of zeros in all but the ith entry, where it is one.

Let $\mathbf{b} \equiv (b_0, b_2, \ldots, b_n)$ be the tuple of barycentric coordinates on a simplex. For a multi-index α, we define $\mathbf{b}^\alpha$ by $\prod_{i=0}^n b_i^\alpha$, which we call a *barycentric monomial*. Scaling these by appropriate generalized binomial coefficients yields the *Bernstein polynomials*. The Bernstein polynomials of degree r have the form

$$B_\alpha^r = \frac{r!}{\alpha!}\mathbf{b}^\alpha. \tag{2.1}$$

For all spatial dimensions and degrees r, the Bernstein polynomials $\{B_\alpha^r\}_{|\alpha|=r}$ form a nonnegative partition of unity and a basis for the space of polynomials of degree r.

To take a partial derivative of a Bernstein polynomial in some direction s, we use the general product rule to write

$$\frac{\partial B_\alpha^r}{\partial s} = \frac{\partial}{\partial s}\left(\frac{r!}{\alpha!}\mathbf{b}^\alpha\right) = \frac{r!}{\alpha!}\sum_{i=0}^n \left(\alpha_i \frac{\partial b_i}{\partial s} b_i^{\alpha_i - 1} \prod_{j=0}^n b_i^{\alpha_i}\right),$$

with the understanding that a term in the sum disappears if $\alpha_i = 0$. This can readily be rewritten as

$$\frac{\partial B_\alpha^r}{\partial s} = r\sum_{i=0}^n B_{\alpha - e_i}^{r-1}\frac{\partial b_i}{\partial s}, \tag{2.2}$$

again with the terms vanishing if any $\alpha_i = 0$, so that the derivative of each Bernstein polynomial is a short linear combination of lower-degree Bernstein polynomials.

Iterating over spatial directions, the gradient of each Bernstein polynomial can be written as

$$\nabla B_\alpha^r = r\sum_{i=0}^n B_{\alpha - e_i}^{r-1}\nabla b_i. \tag{2.3}$$

Note that each ∇b_i is a fixed vector in $\mathbb{R}^n$ for a given simplex T. For a collection of cells, the gradients of Bernstein polynomials are formed as exactly the same pattern of lower-degree Bernstein polynomials times these fixed vectors—the pattern remains the same, but the numerical values of the vectors change between cells. In this way,

one may store the pattern needed to form the linear combinations once and then store the numerical values of ∇b_i in an $N_c \times n$ array, where N_c is the number of simplices in the mesh.

2.2. An example. We flesh out this approach with an example. Consider $r = n = 2$, quadratic Bernstein polynomials on a triangle. We use the lexicographic ordering to enumerate the Bernstein polynomials as

$$\begin{aligned} \beta_0^2 &\equiv B^2_{(0,0,2)}, \\ \beta_1^2 &\equiv B^2_{(0,1,1)}, \\ \beta_2^2 &\equiv B^2_{(0,2,0)}, \\ \beta_3^2 &\equiv B^2_{(1,0,1)}, \\ \beta_4^2 &\equiv B^2_{(1,1,0)}, \\ \beta_5^2 &\equiv B^2_{(2,0,0)}. \end{aligned} \tag{2.4}$$

Since the gradients of these will be linear polynomials, we also enumerate the linear Bernstein polynomials by

$$\begin{aligned} \beta_0^1 &\equiv B^1_{(0,0,1)}, \\ \beta_1^1 &\equiv B^1_{(0,1,0)}, \\ \beta_2^1 &\equiv B^1_{(1,0,0)}. \end{aligned} \tag{2.5}$$

Using (2.3), the gradient of $\beta_0 = B^2_{(0,0,2)}$ is

$$\nabla \beta_0^2 = \nabla B^2_{(0,0,2)} = 2B^1_{(0,0,1)} \nabla b_2. = 2\beta_0^1 \nabla b_2.$$

Similarly, we calculate that

$$\begin{aligned} \nabla \beta_1^2 &= 2\beta_0^1 \nabla b_1 + 2\beta_1^1 \nabla b_2, \\ \nabla \beta_2^2 &= 2\beta_1^1 \nabla b_1, \\ \nabla \beta_3^2 &= 2\beta_0^1 \nabla b_0 + 2\beta_2^1 \nabla b_2, \\ \nabla \beta_4^2 &= 2\beta_1^1 \nabla b_0 + 2\beta_2^1 \nabla b_1, \\ \nabla \beta_5^2 &= 2\beta_2^1 \nabla b_0. \end{aligned}$$

We can specify the structure of these gradients for all triangles with some basic arrays. Each gradient consists of one or more terms of the form of a scalar times a linear Bernstein polynomial times the gradient of a barycentric coordinate. (In this case, all the scalars are 2, but we will need a more general structure.)

We have a total of six quadratic Bernstein polynomials, and there are a total of nine linear terms in the expansion of these six gradients. We introduce several arrays that store this information. Each Bernstein polynomial in the expansion of each gradient is scaled by the number 2. We define S to be the array consisting of the number 2 repeated nine times:

$$\mathrm{S} = [2, 2, 2, 2, 2, 2, 2, 2, 2].$$

We also store the arrays

$$\mathrm{L} = [0, 0, 1, 1, 0, 2, 1, 2, 2]$$

and

$$\mathrm{G} = [2, 1, 2, 1, 0, 2, 0, 1, 0]$$

that contain the indices of the linear Bernstein polynomials and barycentric gradients appearing in the linear combinations, respectively. These are obtained by concatenating the indices of the linear Bernstein polynomials and Bernstein gradients that appear in the linear combinations above—$\nabla\beta_0^2$ is 2 times the β_0^1 times the gradient of b_2, so our first entries of S, L, and G are 2, 0, and 2, respectively. The next gradient $\nabla\beta_1^2 = 2\beta_0^1\nabla b_1 + 2\beta_1^1\nabla b_2$ consists of two terms. Both scalars are 2, so the second and third entries in S are both 2. The 0 and 1 in the second and third entries of L correspond to the bottom indices of β_0^1 and β_1^1. Since the gradients of b_1 and b_2 appear in the terms, the second and third entries of G must be 1 and 2. Going down the list of barycentric gradients in the same manner fills in the rest of these entries.

One more data structure is needed to demarcate the beginnings of each basis function in S, L, and R. To this end, we introduce the array

$$\mathrm{R} = [0, 1, 3, 4, 6, 8, 9].$$

R[0] is 0, so the information for the first basis function begins in entries 0 of S, L, and G. Since R[1] − R[0] = 1, we know there is only one term to build the first gradient. The second gradient starts at entry 1 of S, L, and G and has two terms (R[2] − R[1] = 2). The length of R is one more than the number of basis gradients, and the final entry of R gives the total number of entries in S, L, and G.

We call this collection a *pattern*. Given values of the linear Bernstein polynomials at a collection of points, it is possible to evaluate the gradients of the quadratic Bernstein polynomials at those points by looping through these data structures and incorporating the particular values of the barycentric gradients, much like multiplying a compressed sparse row matrix onto a vector.

2.3. Stroud conical rules and the Duffy transform. The Duffy transform [10] tensorializes the Bernstein polynomials, so sum factorization can be used for evaluating and integrating these polynomials with Stroud conical quadrature. We used similar quadrature rules in our own work on Bernstein–Vandermonde–Gauss matrices [16], but the connection to the Duffy transform and decomposition of Bernstein polynomials was quite cleanly presented by Ainsworth, Andriamaro, and Davydov in [1].

The Duffy transform maps any point $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ in the n-cube $[0, 1]^n$ into the barycentric coordinates for an n-simplex by first defining

$$\lambda_0 = t_1, \tag{2.6}$$

then inductively

$$\lambda_i = t_{i+1}\left(1 - \sum_{j=0}^{i-1}\lambda_j\right) \tag{2.7}$$

for $1 \le i \le n-1$, and then finally

$$\lambda_n = 1 - \sum_{j=0}^{n-1}\lambda_j. \tag{2.8}$$

If a simplex T has vertices $\{\mathbf{x}_i\}_{i=0}^n$, then the mapping

$$\mathbf{x}(\mathbf{t}) = \sum_{i=0}^{n} \mathbf{x}_i \lambda_i(\mathbf{t}) \tag{2.9}$$

maps the unit n-cube onto T.

This mapping can be used to write integrals over T as iterated weighted integrals over $[0,1]^n$:

$$\int_T f(\mathbf{x}) d\mathbf{x} = \frac{|T|}{n!} \int_0^1 dt_1 (1-t_1)^{n-1} \int_0^1 dt_2 (1-t_2)^{n-2} \dots \int_0^1 dt_t f(\mathbf{x}(t)). \tag{2.10}$$

The *Stroud conical rule* [32] is based on this observation and consists of tensor products of certain Gauss–Jacobi quadrature weights in each t_i variable, where the weights are chosen to absorb the factors of $(1-t_i)^{n-i}$. These rules play an important role in the collapsed-coordinate framework of [13] among many other places.

Vital to fast Bernstein-basis finite element algorithms is that the Duffy transform tensorializes the Bernstein polynomials. In [1], it is shown that with $B_i^r(t) = \binom{r}{i} t^i (1-t)^{r-i}$ the one-dimensional Bernstein polynomial, it holds that

$$B_\alpha^r(\mathbf{x}(\mathbf{t})) = B_{\alpha_0}^r(t_1) B_{\alpha_1}^{r-\alpha_0}(t_2) \cdots B_{\alpha_{n-1}}^{r-\sum_{i=0}^{n-2}\alpha_i}(t_n). \tag{2.11}$$

2.4. Basic algorithms. The Stroud conical rule and tensorialization of Bernstein polynomials under the Duffy transform lead to highly efficient algorithms for evaluating B-form polynomials and approximating moments of functions against sets of Bernstein polynomials.

Three algorithms based on this decomposition turn out to be fundamental for optimal assembly and application of Bernstein-basis bilinear forms. First, any polynomial $u(\mathbf{x}) = \sum_{|\alpha|=r} \mathrm{u}_\alpha B_\alpha^r(\mathbf{x})$ may be evaluated at the Stroud conical points in $\mathcal{O}(r^{n+1})$ operations. In [16], this result is presented as exploiting a certain block structure in the matrix tabulating the Bernstein polynomials at quadrature points. In [1], it is done by explicitly factoring the sums.

Second, given some function $f(\mathbf{x})$ tabulated at the Stroud points, it is possible to approximate the set of Bernstein moments

$$\mu_\alpha^r(f) = \int_T f(\mathbf{x}) B_\alpha^r d\mathbf{x}$$

for all $|\alpha| = r$ via Stroud quadrature in $\mathcal{O}(r^{n+1})$ operations. In the case where f is constant on T, we may also use the algorithm for applying a mass matrix in [15] to bypass numerical integration.

Finally, it is shown in [1] that the moment calculation can be adapted to the evaluation of element mass and hence stiffness and convection matrices by utilizing another remarkable property of the Bernstein polynomials. Namely, the product of two Bernstein polynomials of any degrees is, up to scaling, a Bernstein polynomial of higher degree:

$$B_\alpha^r B_\beta^s = \frac{\binom{\alpha+\beta}{\alpha}}{\binom{r+s}{r}} B_{\alpha+\beta}^{r+s}. \tag{2.12}$$

This fact leads to optimal-complexity assembly of element matrices. For a basis consisting of $\mathcal{O}(r^n)$ members, the element matrix contains $\mathcal{O}(r^{2n})$ entries. To evaluate

the variable coefficient mass matrix

$$M_{\alpha\beta} = \int_T c(\mathbf{x}) B^r_\alpha(\mathbf{x}) B^s_\beta(\mathbf{x}) d\mathbf{x}, \tag{2.13}$$

one writes

$$\begin{aligned} M_{\alpha\beta} &= \int_T c(\mathbf{x}) \frac{\binom{\alpha+\beta}{\alpha}}{\binom{r+s}{r}} B^{r+s}_{\alpha+\beta} d\mathbf{x} \\ &= \frac{\binom{\alpha+\beta}{\alpha}}{\binom{r+s}{r}} \mu^{r+s}_{\alpha+\beta}(c). \end{aligned} \tag{2.14}$$

To efficiently form M, one first computes all of the degree $r+s$ moments of the weight function $c(\mathbf{x})$:

$$\left\{ \mu^{r+s}_\alpha(c) \right\}_{|\alpha|=r+s}.$$

Supposing $r = s$, this means evaluating all moments of degree $2r$, which costs about 2^{n+1} times the cost of forming all moments of degree r, or is itself an $\mathcal{O}(r^{n+1})$ process. After the moments are obtained, forming the entire mass matrix consists of scaling these moments by the appropriate binomial coefficients, which requires some care.

This optimal assembly algorithm makes heavy use of the fact that, up to scaling, the mass matrix contains many repeated entries. We remark in [16] that the one-dimensional mass matrix is always a row and column scaling of a Hankel matrix and that the blocks of a higher-dimensional mass matrix consist of scaled lower-dimensional mass matrices. Since this fact persists with variable coefficients, Ainsworth's optimal assembly is a more general use of this fact.

Also, the first two algorithms described above for evaluation and moment calculations demonstrate that M may be applied to a vector without explicitly forming its entries in only $\mathcal{O}(r^{n+1})$ entries.

Now these algorithms can be readily adapted to evaluate matrices whose entries consist of short linear combinations of Bernstein polynomials. This is true whether the coefficients in those linear combinations are scalars, vectors, or even tensors of some sort. Since any directional derivative of each B-form polynomial is such a linear combination with scalar coefficients and the gradient a linear combination with vector coefficients, our technique here recreates the stiffness and convection matrix discussion in [1].

Suppose we have a collection of functions $\{\chi_i\}_{i=1}^{\mathcal{N}}$ of the form

$$\chi_i = \sum_{j=1}^{m_i} s^i_j a^i_j B^r_{\alpha^i_j}. \tag{2.15}$$

That is, each χ_i is the sum over some m_i terms of the form scalar times Bernstein polynomial times a vector $a^i_j \in \mathbb{R}^N$. We will assume that each $m_i < m$ for some small m.

Now, for some function $c(\mathbf{x})$, define the $\mathcal{N} \times \mathcal{N}$ matrix

$$K_{ij} = \int_T c(\mathbf{x}) \chi_i \cdot \chi_j d\mathbf{x}. \tag{2.16}$$

Expanding the definition of χ_i, we have

$$\begin{aligned} K_{ij} &= \int_T c(\mathbf{x})\chi_i \cdot \chi_j d\mathbf{x} \\ &= \int_T c(\mathbf{x}) \left(\sum_{k=1}^{m_i} s_k^i a_k^i B^r_{\alpha_k^i} \right) \cdot \left(\sum_{\ell=1}^{m_j} s_\ell^j a_\ell^j B^r_{\alpha_\ell^j} \right) d\mathbf{x} \\ &= \sum_{k=1}^{m_i} s_k^i s_\ell^j \sum_{\ell=1}^{m_j} \left(a_k^i \cdot a_\ell^j \right) \left(\int_T c(\mathbf{x}) B^r_{\alpha_k^i} B^r_{\alpha_\ell^j} d\mathbf{x} \right) \\ &= \sum_{k=1}^{m_i} \sum_{\ell=1}^{m_j} \left(a_k^i \cdot a_\ell^j \right) M_{\alpha_k^i \beta_\ell^j}. \end{aligned} \tag{2.17}$$

We define the rank-four tensor $A_{k\ell}^{ij}$ by

$$A_{k\ell}^{ij} = s_k^i s_\ell^j \left(a_k^i \cdot a_\ell^j \right) \tag{2.18}$$

and the logical submatrix $\tilde{M}$ of M by

$$\tilde{M}_{k\ell}^{ij} = M_{\alpha_k^i \alpha_\ell^j}. \tag{2.19}$$

With these two definitions, we rewrite the entries of K as

$$K_{ij} = \sum_{k=1}^{m_i} \sum_{\ell=1}^{m_j} A_{k\ell}^{ij} \tilde{M}_{k\ell}^{ij}. \tag{2.20}$$

We can compute K with optimal-order complexity by first forming the moments $\left\{ \mu_\alpha^{2r}(c) \right\}_{|\alpha|=2r}$. Then, for each $A_{k\ell}^{ij}$, we require a dot product costing N multiply-add pairs and scaling $s_k^i s_\ell^j$ costing two more multiplications. So, forming all $A_{k\ell}^{ij}$ costs $m_i m_j (N+2)$ flops, assuming a fused multiply-add. We then form each $\tilde{M}_{k\ell}^{ij}$ by scaling each moment $\mu^{2n}_{\alpha_k^i + \alpha_\ell^j}(c)$ by a ratio of binomial coefficients

$$\frac{\binom{\alpha_k^i + \alpha_\ell^j}{\alpha_k^i}}{\binom{2n}{n}},$$

requiring an additional $m_i m_j$ multiplications. The final entry K_{ij} follows by contracting $A_{k\ell}^{ij}$ and $\tilde{M}_{k\ell}^{ij}$, requiring $m_i m_j$ more multiply-add pairs. This means that the total cost of building K is the cost of moment formation plus an additional $m_i m_j (N+4)$ flops per entry. This operation count remains of constant order per matrix entry.

In light of (2.3), then, this approach can be used to assemble the element stiffness matrix

$$K_{\alpha\beta} = \int_T c(\mathbf{x}) \nabla B_\alpha^r \cdot \nabla B_\beta^r d\mathbf{x} \tag{2.21}$$

at the cost of assembling all Bernstein moments $\mu_\alpha^{2r-2}(c)$ plus no more than an additional $(n+1)^2 (2+n)$ operations per entry.

This approach also allows optimal computation of moments of some function $\underset{\sim}{f} : T \to \mathbb{R}^N$ against all $\{\chi_i\}_{i=1}^{\mathcal{N}}$. Let $\{f^\ell\}_{\ell=1}^N$ with $f^\ell : T \to \mathbb{R}$ be the individual

components of the vector-valued function $\underset{\sim}{f}$. Similarly, let $\{a^i_{j,\ell}\}_{\ell=1}^N$ denote the N components of each a^i_j. Then,

$$\begin{aligned}\int_T \underset{\sim}{f}(\mathbf{x}) \cdot \chi_i d\mathbf{x} &= \int_T \underset{\sim}{f}(\mathbf{x}) \cdot \sum_{j=1}^{m_i} s^i_j a^i_j B^r_{\alpha^i_j} d\mathbf{x} \\ &= \int_T \sum_{j=1}^{m_i} \sum_{\ell=1}^{N} s^i_j a^i_{j,\ell} f^\ell(\mathbf{x}) B^r_{\alpha^i_j} d\mathbf{x} \\ &= \sum_{j=1}^{m_i} s^i_j \sum_{\ell=1}^{N} a^i_{j,\ell} \int_T f^\ell(\mathbf{x}) B^r_{\alpha^i_j} d\mathbf{x}.\end{aligned} \tag{2.22}$$

To evaluate this, we first calculate the N sets of moments $\{\mu^r_\alpha(f^\ell)\}_{\ell=1}^N$. Afterward, the desired moments against $\{\chi_i\}$ are found by more arithmetic. The innermost sum above costs N flops (a dot product of a^i_j with a subset of the moments), with another multiplication to scale by s^i_j. There are m_i such computations, for no more than $m\,(N+1)$ flops per moment against χ_i calculated.

Before proceeding, this discussion suggests a great deal of reuse in a computer program. Once the evaluation, integration, and moment routines are in place for Bernstein polynomials, a small amount of additional code combines these routines with the patterns for $\{\chi_i\}$ to give general, optimal-complexity algorithms. This works very well for constant- or variable-coefficient H^1 stiffness matrices, and we shall also show soon that it works for the rest of the de Rham complex.

3. Finite element exterior calculus bases. The finite element exterior calculus [5] has provided a unified explanation of the theoretical properties of mixed finite element spaces developed over the past several decades. For our purposes, we are interested in the description of suitable elementwise bases found in [6] for the de Rham complex. By working in terms of the local bases, we are able to give a more general approach than by handling the $H(\mathrm{div})$ and $H(\mathrm{curl})$ bases separately.

For a domain $\Omega \subset \mathbb{R}^d$, we let $H\Lambda^k(\Omega)$ denote the space of all differential k-forms ω in L^2 whose exterior derivatives $d\omega$ also are in L^2. As subspaces of these spaces, we consider polynomial-based $\Lambda^k_h \subset H\Lambda^k$ such that appropriate projections into these spaces commute with the exterior derivatives, giving discrete de Rham complexes.

The domain Ω will be tessellated into $\mathcal{T}$, a collection of simplices that is a *triangulation* [8], and we will employ suitable local polynomial spaces. The exterior calculus deals with two such kinds of spaces: $\mathcal{P}_r\Lambda^k(\mathcal{T})$, whose members restricted to any $T \in \mathcal{T}$ are polynomial k-forms of degree r, and the somewhat smaller space $\mathcal{P}^-_r\Lambda^k(\mathcal{T})$, which sits somewhere between polynomials of degree $r-1$ and degree r. These spaces correspond to the classical second and first kind spaces developed by Nédélec [24, 25], respectively.

Following the notation developed in [6], we use increasing maps on the integers to index simplicial facets, Bernstein polynomials, and associated differential forms. Let j, k, l, m be integers with $0 \le k - j \le m - l$ and let $\Sigma(j:k, l:m)$ be the set of all increasing maps from the integers $\{j, \ldots, k\}$ into the integers $\{l, \ldots, m\}$. That is, each $\sigma \in \Sigma(j:k, l:m)$ satisfies $\sigma : \{j, \ldots, k\} \to \{l, \ldots, m\}$ with $\sigma(i) < \sigma(i)$ for each $j \le i < k$.

For $\sigma \in \Sigma(j:k, l:m)$, denote by $[\![\sigma]\!]$ the *range* of σ. That is, $[\![\sigma]\!] = \{\sigma(i) : j \le i \le k\}$. For a sequence $\sigma \in \Sigma(0:k, 0:n)$, we will also need the complementary sequence $\sigma^* \in \Sigma(k+1:n, 0:n)$ such that $[\![\sigma]\!] \cap [\![\sigma^*]\!]$ is empty and $[\![\sigma]\!] \cup [\![\sigma^*]\!] = \{0, 1, \ldots, n\}$.

For a multi-index α and increasing sequence σ, we also define $[\![\alpha, \sigma]\!]$ by

$$[\![\alpha, \sigma]\!] = \operatorname{supp} \alpha \cup [\![\sigma]\!],$$

where the support of a multi-index α is the set of indices i for which $\alpha_i \neq 0$.

Now, for a simplex T with vertices $\{\mathbf{x}_i\}_{i=0}^n$, we let $\Delta(T)$ denote all the facets (subsimplices) of T and let $\Delta_k(T)$ denote the subsimplices of dimension k. So, $\Delta_k(T)$ has $\binom{n+1}{k+1}$ elements. The elements of $\Delta(T)$ may be described by increasing sequences. For $\sigma \in \Sigma(j : k, 0 : n)$, we let $f_\sigma \in \Delta(T)$ be the closed convex hull of vertices $\{\mathbf{x}_{\sigma(j)}, \ldots, \mathbf{x}_{\sigma(k)}\}$. In particular, there is a natural one-to-one mapping between $\Sigma(0 : k, 0 : n)$ and $\Delta_k(T)$. For example, if we label a tetrahedron with vertices (x_0, x_1, x_2, x_3), then Δ_1 denotes the edges, which we label $(0, 1)$, $(0, 2)$, $(0, 3)$, $(1, 2)$, $(1, 3)$, and $(2, 3)$. We define the index set $\mathcal{I}(f)$ of a facet f (ordered sequence) by the set of vertex labels defining it.

3.1. $P_r^- \Lambda^k$ spaces. The exterior calculus bases for the $P_r^- \Lambda^k$ spaces employ the Whitney forms ϕ_σ^f associated with each facet $f \in \Delta_k(T)$. These are defined by

$$\phi_\sigma^f = \sum_{i=0}^{k} (-1)^i \, b_{\sigma(i)}^f \bigwedge_{\ell=0}^{k} db_{\sigma(\ell)}^f, \tag{3.1}$$

where the b^f are the barycentric coordinates of the facet f. A key result of [6] is that these functions can be consistently extended from the facet to the whole simplex in such a way that the trace on the facet's complement vanishes.

In fact, the basis functions for $P_r^- \Lambda^k(T)$ associated with a facet f of a simplex T are given by

$$\begin{aligned} \big\{ \mathbf{b}^\alpha \phi_\sigma^T : \alpha \in \mathbb{N}_0^{0:n}, |\alpha| = r - 1, \sigma \in \Sigma(0 : k, 0 : n), [\![\alpha, \sigma]\!] = \mathcal{I}(f), \\ \alpha_i = 0 \text{ if } i < \min [\![\sigma]\!] \big\}. \end{aligned} \tag{3.2}$$

We make a slight modification, replacing the barycentric monomial $\mathbf{b}^\alpha$ with the Bernstein polynomial B_α^r. This gives the set of functions

$$\begin{aligned} \big\{ B_\alpha^{r-1} \phi_\sigma^T : \alpha \in \mathbb{N}_0^{0:n}, |\alpha| = r - 1, \sigma \in \Sigma(0 : k, 0 : n), [\![\alpha, \sigma]\!] = \mathcal{I}(f), \\ \alpha_i = 0 \text{ if } i < \min [\![\sigma]\!] \big\}. \end{aligned} \tag{3.3}$$

Rather than summarizing the derivation given in [6], we give an example below of how this basis may be converted to B-form, much like we did with the Bernstein gradients.

3.2. $P_r \Lambda^k$ spaces. We also consider the bases for $P_r \Lambda^k$. These are given by Arnold, Falk, and Winther, after substituting Bernstein polynomials for the barycentric monomials, as

$$\begin{aligned} \big\{ B_\alpha^r \psi_\sigma^{\alpha, f, T} : \alpha \in \mathbb{N}_0^{0:n}, |\alpha| = r, \sigma \in \Sigma(1 : k, 0 : n), [\![\alpha, \sigma]\!] = \mathcal{I}(f), \\ \alpha_i = 0 \text{ if } i < \min(\mathcal{I}(f) \backslash [\![\sigma]\!]) \big\}, \end{aligned} \tag{3.4}$$

where the vectors $\psi_\sigma^{\alpha, f, T}$ are defined by

$$\psi_\sigma^{\alpha, f, T} = \psi_{\sigma(1)}^{\alpha, f, T} \wedge \cdots \wedge \psi_{\sigma(k)}^{\alpha, f, T}, \quad \sigma \in \Sigma(1 : k, 0 : n), \quad [\![\sigma]\!] \subseteq \mathcal{I}(f), \tag{3.5}$$

with

$$\psi_i^{\alpha,f,T} = db_i^T - \frac{\alpha_i}{|\alpha|}\sum_{j\in\mathcal{I}(f)} db_j^g, \quad i \in \mathcal{I}(f). \tag{3.6}$$

This definition, though somewhat complicated, was required to get the facet basis functions to have vanishing traces off the facet. For our computational purposes, we note that each $\psi_\sigma^{\alpha,f,T}$ is a fixed, geometric vector.

3.3. Basis conversion. Both $P_r^-\Lambda^k$ and $P_r\Lambda^k$ have natural representations as vectors of length $\binom{n}{k}$ whose components are polynomials of degree r.

For each particular r, k, T, we let $\{\xi_j\}_{j=1}^{\dim P_r^-\Lambda^k(T)}$ denote the basis for $P_r^-\Lambda^k(T)$. Each k-form is naturally represented as a vector in $\mathbb{R}^{\binom{n}{k}}$, so we can represent each basis function as a vector of polynomials. In particular, we will expand each vector component in the Bernstein basis. That is, for each i, we seek a collection of vectors $\{a_j^i\} \subset \mathbb{R}^{\binom{n}{k}}$ and indices α^{j_i} with $|\alpha^{j_i}| = r$ such that

$$\xi_i = \sum_j a_j^i B_{\alpha^{j_i}}, \tag{3.7}$$

and a similar expansion for the basis θ_j.

For $P_r^-\Lambda^k$ spaces, each element of the bases in (3.3) has the form of a Bernstein polynomial times a Whitney form. Each Whitney form is in turn a sum of barycentric coordinates times certain geometric vectors. These vectors are the gradients of the barycentric coordinates for 1-forms and wedge (cross) products of these gradients for 2-forms. A generic 1-form basis function has the form $B_\alpha^{r-1}\phi_{ij}$ for some $|\alpha| = r-1$ and some $\sigma = (i,j)$. Expanding the Whitney form, we have

$$\begin{aligned} B_\alpha^{r-1}\phi_{ij} &= B_\alpha^{r-1}\left(b_i db_j - b_j db_i\right) \\ &= b_i B_\alpha^{r-1} db_j - b_j B_\alpha^{r-1} d\lambda_i. \end{aligned} \tag{3.8}$$

Now, the factors $b_i B_\alpha^{r-1}$ are scaled Bernstein polynomials of degree r. To see this,

$$\begin{aligned} b_i B_\alpha^{r-1} &= b_i \frac{(r-1)!}{\alpha!} b^\alpha \\ &= \frac{(\alpha_i+1)}{r}\frac{r!}{(\alpha+e_i)!} b^{\alpha+e_i} \\ &= \frac{(\alpha_i+1)}{r} B_{\alpha+e_i}^r. \end{aligned} \tag{3.9}$$

This calculation renders our typical 1-form basis function as

$$B_\alpha^{r-1}\phi_{ij} = \frac{\alpha_i+1}{r} B_{\alpha+e_i}^r db_j - \frac{\alpha_j+1}{r} B_{\alpha+e_j}^r db_i. \tag{3.10}$$

The same calculation generalizes readily to k-forms.

THEOREM 3.1. *For any $|\alpha| = r-1$ and $\sigma \in \Sigma(0:k, 0:n)$, we have that*

$$B_\alpha^{r-1}\phi_\sigma = \sum_{i=0}^{k} (-1)^i \left(\frac{\alpha_{\sigma(i)}+1}{r}\right) B_{\alpha+e_{\sigma(i)}}^r \bigwedge_{\substack{\ell=0 \\ \ell\neq i}}^{k} db_{\sigma(\ell)}. \tag{3.11}$$

Each k-form basis function for $P_r^-\Lambda^k$, then, is a sum of $k+1$ terms, each of which is a scalar times a Bernstein polynomial times a cell-dependent geometric vector. In

this case, it is a wedge product of k exterior derivatives (gradients) of the barycentric coordinates. Once the barycentric coordinates are differentiated, one can iterate over k-tuples of them and enumerate all of the wedge products needed for each cell, and build a pattern just as we did for the Bernstein gradients. Note that for each cell, there is a small number of geometric vectors, $\binom{n+1}{k+1}$, independent of the polynomial degree r. These vectors, wedge products of exterior derivatives (gradients) of the barycentric coordinates, vary from cell to cell, but the particular pattern remains the same on each cell. In particular, the same CSR-like pattern technique described for Bernstein gradients can be used to store the conversion from the exterior calculus basis to the Bernstein form.

Now we pause to give an example of the pattern obtained by converting the basis functions to B-form. We consider $P_2^-\Lambda^1$ on a triangle. We order the edges by $(1,2)$, $(0,2)$, and then $(0,1)$. We need to enumerate the two basis functions for each edge and then the two internal basis functions and apply (3.8) to each.

Consider the first edge, $(1,2)$. In (3.3), we require that all α and σ associated with this edge have $|\alpha| = 1$ and $[\![\alpha, \sigma]\!] = \{1,2\}$. With $\sigma = (1,2)$, this gives $\alpha = (0,0,1)$ or $\alpha = (0,1,0)$ as the possibilities. Using (3.8) to expand the Whitney forms and (2.4) to use the linear ordering of the quadratic Bernstein polynomials gives

$$\begin{aligned} B^1_{(0,0,1)}\sigma_{(1,2)} &= \frac{1}{2}B^2_{(0,1,1)}db_2 - B^2_{(0,0,2)}db_1 = \frac{1}{2}\beta^2_1 db_2 - \beta^2_0 db_1, \\ B^1_{(0,1,0)}\sigma_{(1,2)} &= B^2_{(0,2,0)}db_2 - \frac{1}{2}B^2_{(0,1,1)}db_1 = \beta^2_2 db_2 - \frac{1}{2}\beta^2_1 db_1. \end{aligned} \tag{3.12}$$

Similarly, for edge $(0,2)$, we have $\sigma = (0,2)$ and $\alpha = (0,0,1)$ or $(1,0,0)$ and find

$$\begin{aligned} B^1_{(0,0,1)}\sigma_{(0,2)} &= \frac{1}{2}B^2_{(1,0,1)}db_2 - B^2_{(0,0,2)}db_0 = \frac{1}{2}\beta^2_3 db_2 - \beta^2_0 db_0, \\ B^1_{(1,0,0)}\sigma_{(0,2)} &= B^2_{(2,0,0)}db_2 - \frac{1}{2}B^2_{(1,0,1)}db_0 = \beta^2_5 db_2 - \frac{1}{2}\beta^2_3 db_0. \end{aligned} \tag{3.13}$$

For edge $(0,1)$ we have $\sigma = (0,1)$ and $\alpha = (0,1,0)$ or $(1,0,0)$ so that

$$\begin{aligned} B^1_{(0,1,0)}\sigma_{(0,1)} &= \frac{1}{2}B^2_{(1,1,0)}db_1 - B^2_{(0,2,0)}db_0 = \frac{1}{2}\beta^2_4 db_1 - \beta^2_2 db_0, \\ B^1_{(1,0,0)}\sigma_{(0,1)} &= B^2_{(2,0,0)}db_1 - \frac{1}{2}B^2_{(1,1,0)}db_0 = \beta^2_5 db_1 - \frac{1}{2}\beta^2_4 db_0. \end{aligned} \tag{3.14}$$

The basis functions associated with the triangle $(0,1,2)$ in (3.3) require a bit more care to enumerate. For each $\sigma = (0,1)$, $(0,2)$, or $(1,2)$ we find the α such that $[\![\alpha, \sigma]\!] = \{0,1,2\}$. When $\sigma = (0,1)$, we require that $\alpha = (0,0,1)$. Likewise, for $\sigma = (0,2)$, $\alpha = (0,1,0)$. However, when $\sigma = (1,2)$ and $\alpha = (1,0,0)$, the condition $\alpha_i = 0$ if $i < \min(\operatorname{supp}\sigma)$ fails to hold. So then, we convert the two internal basis functions to B-form as before:

$$\begin{aligned} B^1_{(0,0,1)}\sigma_{(0,1)} &= \frac{1}{2}B^2_{(1,0,1)}db_1 - \frac{1}{2}B^2_{(0,1,1)}db_0 = \frac{1}{2}\beta^2_3 db_1 - \frac{1}{2}\beta^2_1 db_0, \\ B^1_{(0,1,0)}\sigma_{(0,2)} &= \frac{1}{2}B^2_{(1,1,0)}db_2 - \frac{1}{2}B^2_{(0,1,1)}db_0 = \frac{1}{2}\beta^2_4 db_2 - \frac{1}{2}\beta^2_1 db_0. \end{aligned} \tag{3.15}$$

We order the basis functions as we have listed them and give the pattern. Since each basis function is a combination of two Bernstein polynomials, the row pointer vector is

$$\mathrm{R} = [0,2,4,6,8,10,12,14].$$

From the expansions above, we start with the first term in the first basis function, then the second term, and move through the basis functions in the order they are enumerated, packing up the scalar coefficients (in this case, all $\pm\frac{1}{2}$ or ± 1) into the array

$$\mathrm{S} = \left[\frac{1}{2}, -1, 1, -\frac{1}{2}, \frac{1}{2}, -1, 1, -\frac{1}{2}, \frac{1}{2}, -1, 1, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right].$$

Similarly, we extract the labels of each Bernstein polynomial to obtain

$$\mathrm{B} = [1, 0, 2, 1, 3, 0, 5, 3, 4, 2, 5, 4, 3, 1, 4, 1].$$

Finally, we collect the indices of each of the barycentric derivatives db_i that occur in each combination with

$$\mathrm{G} = [2, 1, 2, 1, 2, 0, 2, 0, 1, 0, 1, 0, 1, 0, 2, 0].$$

This information encodes the conversion of the exterior calculus basis to B-form. An important piece of our implementation is a collection of functions that finds the pattern for any order of differential form for any polynomial degree in any spatial dimension.

For $P_r\Lambda^k$, we have noted that each basis function is only a single Bernstein polynomial times some geometric vector. So, specifying the row pointer, scalars, and Bernstein indices is quite straightforward. On the other hand, each basis function has a different geometric vector so that the cost of storing all of the geometric vectors will be quite high as the polynomial degree increases. While our approach is applicable in this case, our implementation has focused on $P_r^-\Lambda^k$ spaces.

To differentiate the $P_r^-\Lambda^k$ basis functions, we could simply differentiate the Bernstein polynomials in each expansion and then combine the two levels of linear combinations. In practice, however, it is preferable to have a representation of the exterior derivatives directly in terms of the Bernstein polynomials. Let us consider the exterior derivative of a typical $P_r^-\Lambda^k$ function. As with the Bernstein gradients, we assume that any terms in sums vanish if they have a negative entry in a multi-index. We use the expansion (3.11) and our Bernstein differentiation formula to find

$$\begin{aligned}
d\left(B_\alpha^{r-1}\phi_\sigma\right) &= d\left(B_\alpha^{r-1}\sum_{i=0}^{k}(-1)^i\, b_{\sigma(i)}\bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)}\right)\\
&= d\left(\sum_{i=0}^{k}(-1)^i\, B_\alpha^{r-1} b_{\sigma(i)}\bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)}\right)\\
&= d\left(\sum_{i=0}^{k}(-1)^i\left(\frac{\alpha_{\sigma(i)}+1}{r}\right) B_{\alpha+e_{\sigma(i)}}^{r}\bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)}\right)\\
&= \sum_{i=0}^{k}(-1)^i\left(\alpha_{\sigma(i)}+1\right)\sum_{\ell=0}^{n} B_{\alpha+e_{\sigma(i)}-e_\ell}^{r-1} db_\ell \wedge \bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)}.
\end{aligned} \tag{3.16}$$

We consider the internal sum over ℓ in three cases. First, if $\ell = \sigma(i)$, then we have that $B^{r-1}_{\alpha+e_{\sigma(i)}-e_\ell} = B^{r-1}_\alpha$ and that

$$db_\ell \wedge \bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)} = db_{\sigma(i)} \wedge \bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)} = (-1)^i \bigwedge_{j=0}^{k} db_{\sigma(j)}$$

using the anticommutativity of the wedge product. Second, if $\ell \in [\![\sigma]\!] \setminus \{\sigma(i)\}$, then the wedge product contains a repeated term $db_{\sigma(j)}$ for some j and hence vanishes. Finally, suppose that $\ell \notin [\![\sigma]\!]$. We define

$$\nu_{\ell,\sigma,i} \equiv \#\{0 \le j \le k : \sigma(j) < \ell\} \tag{3.17}$$

and let $\varsigma_{\ell,\sigma,i}$ be the unique increasing sequence in $\Sigma(0:k, 0:n)$ such that

$$[\![\varsigma_{\ell,\sigma,i}]\!] = \{\ell\} \cup ([\![\sigma]\!] \setminus \{\sigma(i)\}). \tag{3.18}$$

Then we have

$$db_\ell \wedge \bigwedge_{\substack{j=0\\ j\neq i}}^{k} db_{\sigma(j)} = (-1)^{\nu_{\ell,\sigma,i}} \bigwedge_{j=0}^{k} db_{\varsigma_{\ell,\sigma,i}(j)}. \tag{3.19}$$

Combining these calculations with (3.16) gives the following theorem.

THEOREM 3.2. *With ν_{ℓ,σ_i} defined by* (3.17) *and $\varsigma_{\ell,\sigma,i}(j)$ defined by* (3.18)*, for any $|\alpha| = r-1$ and increasing sequence $\sigma \in \Sigma(0:k, 0:n)$, the exterior derivative of $B^{r-1}_\alpha \phi_\sigma$ satisfies*

$$\begin{aligned} dB^{r-1}_\alpha \phi_\sigma = &\sum_{i=0}^{k} \sum_{\substack{\ell=0\\ \ell\notin[\![\sigma]\!]}}^{n} (-1)^{i+\nu_{\ell,\sigma_i}} \left(\alpha_{\sigma(i)}+1\right) B_{\alpha+e_{\sigma(i)}-e_\ell} \bigwedge_{j=0}^{k} db_{\varsigma_{\ell,\sigma,i}(j)} \\ &+ \left(\sum_{i=0}^{k} \left(\alpha_{\sigma(i)}+1\right)\right) B^{r-1}_\alpha \bigwedge_{j=0}^{k} db_{\sigma(j)}. \end{aligned} \tag{3.20}$$

While this formula is somewhat more complicated than the expansion for the basis function itself, it is still manageable and a short linear combination. The first term, with a sum over i and ℓ, contains $k+1$ times $(n+1-(k+1))$ terms of the form of a scalar times a Bernstein polynomial times some geometric vector. The second term is only one additional term with scalar coefficient $\sum_{i=0}^{k} \alpha_{\sigma(i)} + 1 = (k+1) + \sum_{i=0}^{k} \alpha_{\sigma(i)}$. This gives a total of $(k+1)(n-k)+1$ Bernstein polynomials in the expansion. The geometric terms on each cell are now the set of all $(k+1)$-way wedge products of the barycentric gradients.

4. Numerical results.

4.1. Some comments on implementation. We began by implementing all our algorithms in Python, using `numpy` arrays as general multidimensional arrays where needed. Our code consists of numerical routines for polynomial evaluation and integration at Stroud points, forming matrices based on linear combinations of polynomials, and so forth. We also provide routines to form the local-to-global mappings

needed for assembly. A novel feature is our implementation of *patterns*, which encode the CSR-like arrays needed to convert Bernstein gradients, $P_r^-\Lambda^k$ basis functions, and their exterior derivatives to B-form. This pattern implementation allows for arbitrary order differential forms, polynomial degrees, and spatial dimensions. We used `scipy` for global sparse matrices.

Because of poor performance of the numerically intensive parts of our code, we ported these into Cython [7,33], annotating the variables with static types. This leads to a considerable speedup over standard Python, although we have not attempted to achieve maximal performance. For example, in several places we use function parameters to outsource computations to dimensionally dependent routines. These are currently passed as Python callable objects rather than function pointers, which is probably a source of some overhead at low polynomial degree.

4.2. Timing results. We consider the cost of assembling the mass and stiffness matrices for 1- and 2-forms in three dimensions. In addition to our Bernstein implementation, we used several components of FEniCS [21]. We used the Python interface to DOLFIN [22] to create triangular and tetrahedral meshes. This interface also exposes a just-in-time interface to the FEniCS Form Compiler `ffc` [18,31], which in turn relies on FIAT [14] to generate reference element basis functions.

Using the `UnitCubeMesh` accessible within the FEniCS package DOLFIN, we constructed a mesh consisting of the unit cube divided into $10\times10\times10$ cubes, each divided into six tetrahedra for a total of 6000 cells. Then, for each polynomial degree from 1–8, we assembled the mass and stiffness matrices for 1- and 2-forms, reporting an average timing over several iterations. For the lowest polynomial degree, the 1-form matrices had 7930 rows and 119530 nonzeros. For degree 5, they had 471650 rows and 103788250 nonzeros, and they increase from there.

As a point of reference, we also compared our times to assemble the matrices with those of FEniCS. Using the nodal bases for the tetrahedral Raviart–Thomas–Nédélec elements available through FIAT, we used the tensor-contraction mode of `ffc`. This gives matrices of exactly the same dimensions as our Bernstein techniques, although with different numerical values. The FEniCS-generated code performs, per cell, a contraction of a reference-element tensor containing precomputed integrals with a geometric tensor that depends on the cell Jacobian. For constant coefficients, this geometric tensor is independent of the polynomial degree or basis. For mass matrices, the geometric tensor is quite cheap to compute, and a mere nine flops per entry of the element matrix are required after its formation. The geometric tensor for the stiffness matrix is a bit more involved, and we refer the reader to [31] for more details. At any rate, all numerical integration is performed once on a reference cell so that (in theory) we should see constant work per entry as the polynomial degree increases. This would not hold, however, if we were to use the quadrature mode developed in [26]. As an additional remark, it is quite expensive to generate the tensor-contraction code for higher degree polynomials. In the worst case, code generation for the degree 8 mass matrix for 1-forms required more than 37 hours. The stiffness matrix code generation was more efficient, requiring more on the order of minutes for higher degrees.

We plot the timings for 1- and 2-form mass and stiffness matrices in Figure 1. With FEniCS, we see a split between the mass and stiffness matrices for both 1- and 2-forms, with mass matrix construction being considerably faster. Our Bernstein algorithms are close to each other in both cases, although we fail to beat the FEniCS mass matrix for any polynomial degree. However, that we are within an order of

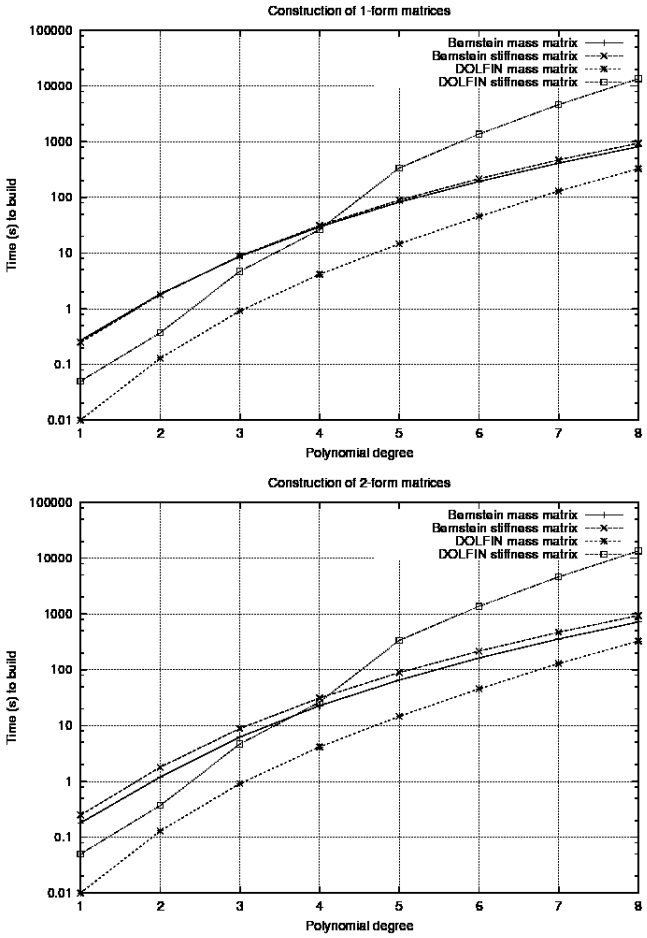

FIG. 1. *Time (in seconds) to build the* 1- *and* 2*-form mass and stiffness matrices using FEniCS and our Bernstein algorithms at degrees* 1–5 *on a mesh of* 6000 *tetrahedra. The FEniCS-based mass matrix assembly greatly outperforms our methods, but the reverse holds for the stiffness matrix after degree* 2 *for* 1*-form matrices and degree* 1 *for* 2*-form matrices.*

magnitude using a quadrature-based method is encouraging for our techniques.

An important feature of the assembly algorithm we used that is also shared by the tensor-contraction mode for constant coefficient problems is that the work for building the element matrices is of constant order per entry as a function of the polynomial degree. In Figure 2, we examine this feature empirically by plotting the assembly time divided by the global nonzeros for each polynomial degree. The FEniCS code does not seem to flatten out, which could be due to how entries are inserted into the global sparse PETSc matrix.

We also considered the cost of matrix-free application of the mass and stiffness matrices to a given vector. In our Bernstein code we used the typical process of scattering the global degrees of freedom to each cell, locally applying the operator, and summing the results back to global storage. This tests the combined performance of our polynomial evaluation, moment calculation, and transformations between the $P_r^-\Lambda^k$ bases and the Bernstein polynomials. For the FEniCS implementation, we used the

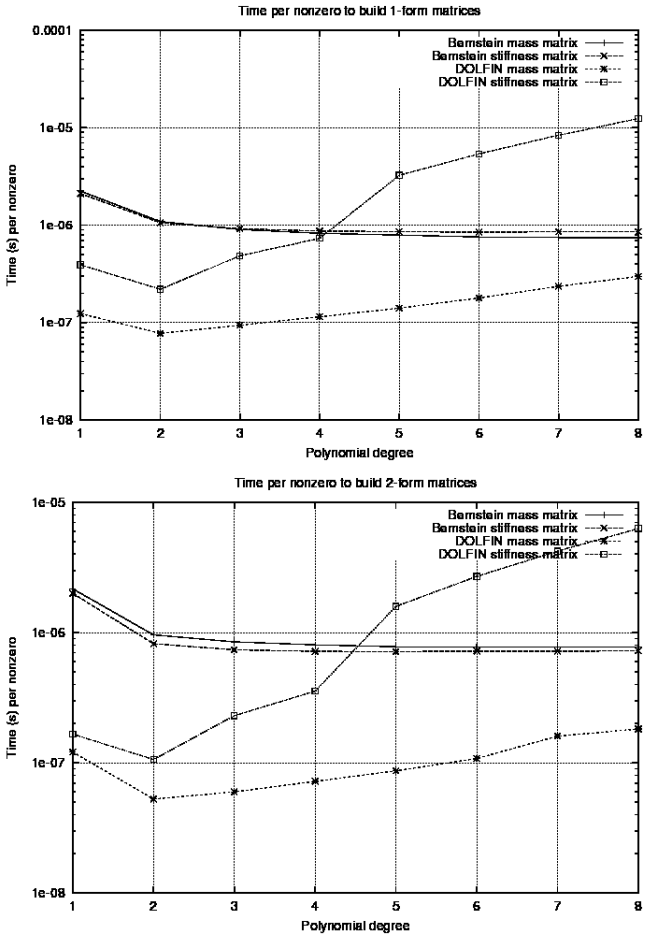

FIG. 2. *Time per nonzero entry to assemble the global mass and stiffness matrices on a mesh of* 6000 *tetrahedra using polynomial orders* 1–5 *for* 1*-form (top) and* 2*-form (bottom) mass and stiffness matrices. With Bernstein polynomial algorithms, the cost per nonzero seems to approach a constant as the degree increases, but this does not hold for the FEniCS-based implementations. Whether this is a feature of the sparse matrix insertion or the element matrix construction is uncertain.

same technique as in [19], where we declare `u` to be a `Function` and `v` a `TestFunction` and assemble the forms `inner(u,v)*dx`, `inner(curl(u),curl(v))*dx`, and `div(u)*div(v)*dx` for the mass and stiffness matrices. In Figure 3, we see that the Bernstein-based algorithms for mass and stiffness matrices both dramatically outperform FEniCS at high order, although FEniCS performs very well at low order.

4.3. Accuracy. Typically, finite element shape functions are chosen to optimize the conditioning and sparsity of element stiffness matrices. The Bernstein polynomials do neither. They give dense element matrices that yet may be constructed and multiplied onto vectors very efficiently. They can also give rise to very poorly conditioned matrices, so we will empirically consider their accuracy at very high order, at least on coarse meshes.

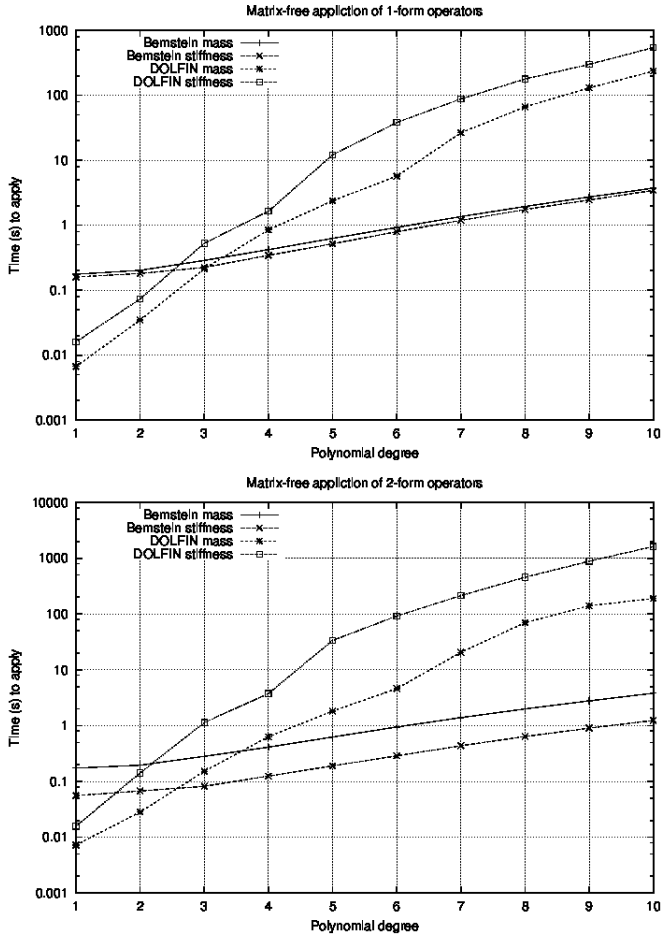

FIG. 3. *Time (in seconds) to perform the matrix-free action of the mass and stiffness matrices for* 1- *and* 2*-forms using Bernstein polynomials and DOLFIN on a mesh of* 6000 *tetrahedra and polynomial degrees* 1–10. *For degrees* 4 *and higher, the Bernstein matrix-free algorithms are significantly faster, although DOLFIN's low-order methods perform very well. The separation between mass and stiffness Bernstein matrices in* 2*-forms results from the relevant geometric vectors having different lengths, whereas they are the same length in* 1*-forms.*

First, we consider the mixed Poisson problem

$$\begin{aligned} \nabla \cdot u &= f, \\ u + \nabla p &= 0 \end{aligned} \tag{4.1}$$

posed on the unit cube with homogeneous Dirichlet boundary conditions. Here u is the flux field sought in $H(\mathrm{div})$ and p is the scalar potential. We pick f such that the true solution is $p(x, y, z) = \frac{3}{\pi^2} \sin(\pi x) \sin(\pi y) \sin(\pi z)$.

To study the accuracy with respect to the polynomial degree r, we subdivide the unit cube into six tetrahedra, again with DOLFIN's `UnitCubeMesh`.

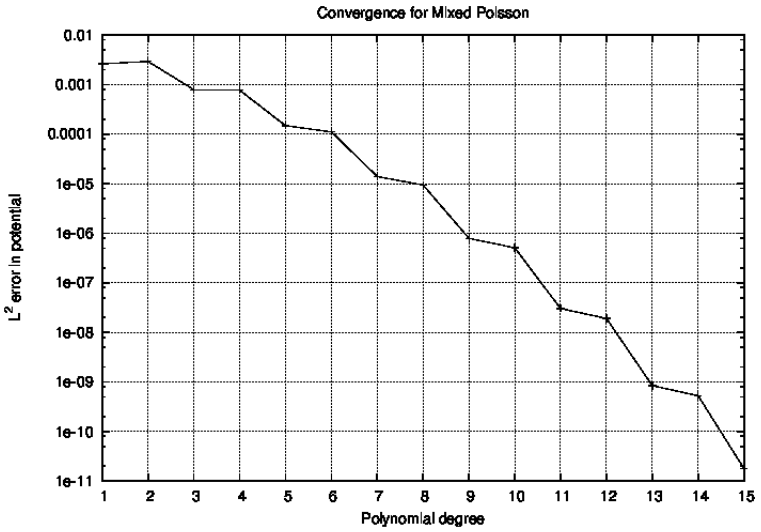

FIG. 4. *Convergence of the mixed finite element method as a function of the polynomial degree on a mesh with six tetrahedra. At degree* 15*, the* L^2 *error has dropped below* 10^{-10}.

We use the standard mixed finite element formulation of $u_h \in V_h$ and $p_h \in W_h$ such that

$$\begin{aligned} (\nabla \cdot u_h, w_h) &= (f, w_h), \\ (u_h, v_h) - (p_h, \nabla \cdot v_h) &= 0 \end{aligned} \tag{4.2}$$

for all $w_h \in W_h$ and $v_h \in V_h$. W_h is the space of discontinuous polynomials of degree $r-1$, and V_h is the Raviart–Thomas space of order $r-1$, or equivalently, $P_r^- \Lambda^2(\mathcal{T})$. Because of the small number of cells, we use dense storage for the mass and divergence matrices and the form and factor the Schur complement using LAPACK via `numpy` [27].

In Figure 4, we see r-convergence—as the polynomial degree increases, the smooth solution is very well approximated. However, the convergence seems somewhat jagged, with small drops from odd to the next-higher even degree followed by larger drops going from even up to odd. To gain more insight, we plotted the error for odd and even degrees separately in Figure 5. In this plot, we see fairly typical exponential convergence plots, although the constants seem different between the two degrees.

It is also interesting to examine the condition number of the resulting system. For each of the system matrices, we also measured the 2-norm condition number as the ratio of the largest to smallest singular value. For our six-cell mesh, we plot this as a function of polynomial degree in Figure 6. The accuracy we obtained in the finite element solution is surprising in light of the very large condition numbers, and we do not have an explanation for this. The condition number provides only a worst-case situation that seems, in this case, quite pessimistic.

Next, we turn to the Maxwell cavity resonator, for which Nédélec elements provide an important tool. We consider the problem of finding resonances $\omega \in \mathbb{R}$ and

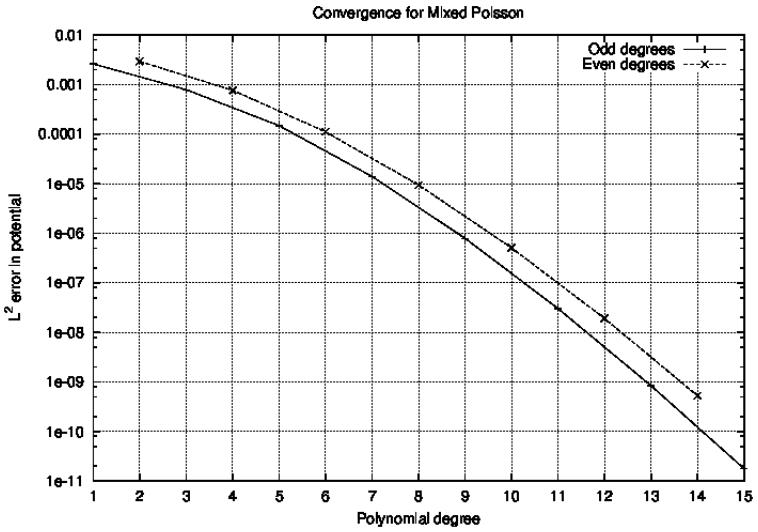

FIG. 5. *Convergence of the mixed finite element as a function of the polynomial degree, separating out the odd and even degrees into separate convergence plots. The degrees of different parity both seem to converge as the degree increases at the same rate, but with different constants.*

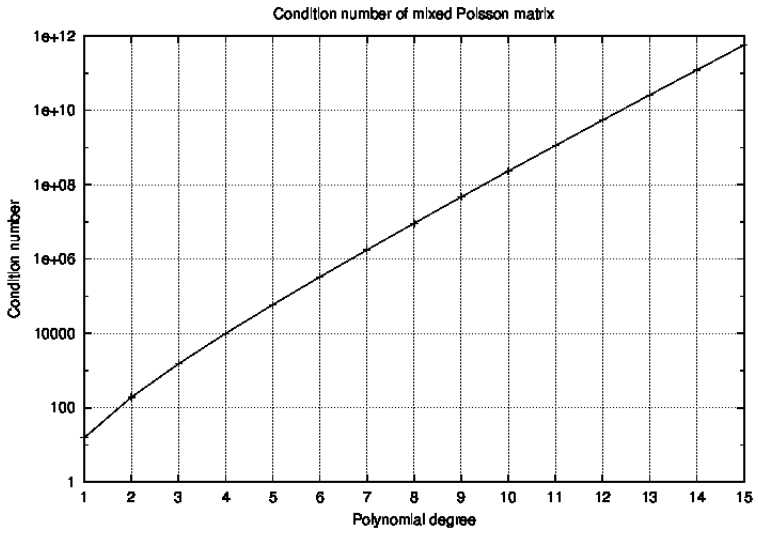

FIG. 6. *Exponential growth of the condition number of mixed finite element stiffness matrix on a six-tetrahedron mesh for polynomial degrees* 1–15.

eigenfunctions $E \in H_0(\mathrm{curl})$ such that

$$(\nabla \times E, \nabla \times F) = \omega^2 (E, F) \tag{4.3}$$

for all $F \in H_0(\mathrm{curl})$. Here, $H_o(\mathrm{curl})$ is the subspace of $H(\mathrm{curl})$ with vanishing tangential component on $\partial\Omega$.

We consider both the square $[0, \pi]^2$ divided into a pair of right triangles and the cube $[0, \pi]^3$ divided into six tetrahedra and study the r-convergence of the finite

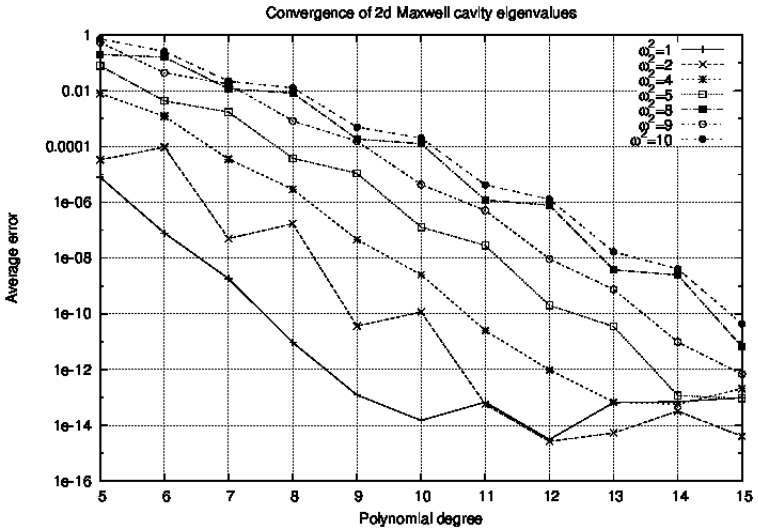

FIG. 7. *Error in the eigenvalues no larger than* 10 *for the two-dimensional cavity resonator problem using polynomials of degrees* 5–15.

element eigenvalue problem. The eigenvalues are known to be integral, consisting of certain sums of squares of nonzero integers [23, 31]. As with the mixed Poisson equation, we use dense storage and use LAPACK to solve the resulting (generalized) eigenvalue problem.

For the two-dimensional problem, the first several eigenvalues are 1, 2, 4, 5, 8, 9, and 10 with respective multiplicities 2, 1, 2, 2, 1, 2, and 2. We used polynomial degrees 5–15 to approximate these eigenvalues on a mesh consisting of two right triangles. Figure 7 shows that at degree 15, all of these eigenvalues are resolved to within 10^{-10}. At the highest degrees, we start to see a slight increase in the error of the smallest eigenvalues, but the results are still very accurate.

For the three-dimensional problem, the first eigenvalues are 2 with multiplicity 3, 3 with multiplicity 2, and 5 with multiplicity 6. We start with polynomial degree 2 since the degree 1 space has only a single degree of freedom corresponding to the internal edge. We take the first 11 computed nonzero eigenvalues and measure the average error in each one. In Figure 8, we see that 10-digit accuracy in all three of the first eigenvalues is obtained when we reach degree 13. As with the mixed Poisson problem, the results seem to exhibit an odd/even separation. No effects of ill-conditioning are apparent, except that the curve for the smallest eigenvalue may be flattening out as the error approaches $\mathcal{O}(10^{-13})$.

5. Conclusions. We have presented the first known optimal-complexity algorithms for forming and applying finite element matrices for the de Rham complex on simplices. These algorithms utilize existing techniques for Bernstein polynomials to give high efficiency and accuracy for the test problems we considered.

In the future, we hope to study how these methods port to GPUs and other accelerators and how they apply to more complicated problems. Efficient solvers and preconditioners for the resulting high-order global linear systems also remain open questions. We are also interested in adapting these techniques to other spline-type spaces [20] with bases constructed from Bernstein polynomials.

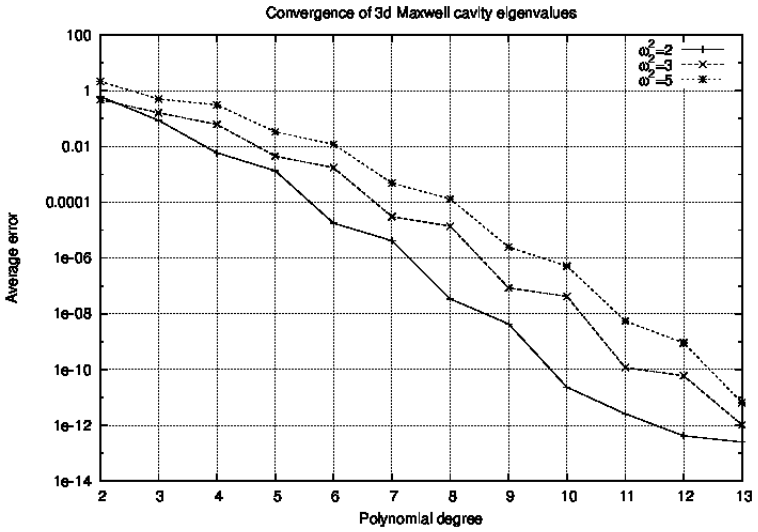

FIG. 8. *Error in the first three distinct eigenvalues for the three-dimensional cavity resonator problem.*

REFERENCES

[1] M. AINSWORTH, G. ANDRIAMARO, AND O. DAVYDOV, *Bernstein–Bézier finite elements of arbitrary order and optimal assembly procedures*, SIAM J. Sci. Comput., 33 (2011), pp. 3087–3109.

[2] M. AINSWORTH, G. ANDRIAMARO, AND O. DAVYDOV, *A Bernstein-Bezier Basis for Arbitrary Order Raviart-Thomas Finite Elements*, Technical report 2012-20, Scientific Computing Group, Brown University, Providence, RI, 2012.

[3] M. AINSWORTH, G. ANDRIAMARO, AND O. DAVYDOV, *A Bernstein-Bezier basis for arbitrary order Raviart-Thomas finite elements*, Constr. Approx., submitted.

[4] D. N. ARNOLD, R. S. FALK, AND R. WINTHER, *Multigrid in H(div) and H(curl)*, Numer. Math., 85 (2000), pp. 197–217.

[5] D. N. ARNOLD, R. S. FALK, AND R. WINTHER, *Finite element exterior calculus, homological techniques, and applications*, Acta Numer., 15 (2006), pp. 1–155.

[6] D. N. ARNOLD, R. S. FALK, AND R. WINTHER, *Geometric decompositions and local bases for spaces of finite element differential forms*, Comput. Methods Appl. Mech. Engrg., 198 (2009), pp. 1660–1672.

[7] S. BEHNEL, R. BRADSHAW, AND G. EWING, *Cython: C-Extensions for Python*, http://cython.org (2008).

[8] S. C. BRENNER AND L. R. SCOTT, *The Mathematical Theory of Finite Element Methods*, 3rd ed., Texts Appl. Math. 15, Springer, New York, 2008.

[9] J. BROWN, *Efficient nonlinear solvers for nodal high-order finite elements in 3D*, J. Sci. Comput., 45 (2010), pp. 48–63.

[10] 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.

[11] J. GOPALAKRISHNAN, L. E. GARCÍA-CASTILLO, AND L. F. DEMKOWICZ, *Nédélec spaces in affine coordinates*, Comput. Math. Appl., 49 (2005), pp. 1285–1294.

[12] B. HIENTZSCH, *Fast solvers and Schwarz preconditioners for spectral Nédélec elements for a model problem in H(curl)*, in Proceedings of the Fourteenth International Conference on Domain Decomposition Methods, I. Herrera, D. E. Keyes, O. B. Widlund, and R. Yates, eds., Natl. Auton. Univ. Mex., Mexico, 2003, pp. 427–433.

[13] G. E. KARNIADAKIS AND S. J. SHERWIN, *Spectral/hp Element Methods for Computational Fluid Dynamics*, 2nd ed., Numer. Math. Sci. Comput., Oxford University Press, New York, 2005.

[14] R. C. KIRBY, *FIAT: A new paradigm for computing finite element basis functions*, ACM Trans. Math. Software, 30 (2004), pp. 502–516.

[15] R. C. KIRBY, *Fast simplicial finite element algorithms using Bernstein polynomials*, Numer. Math., 117 (2011), pp. 631–652.

[16] R. C. Kirby and T. T. Kieu, *Fast simplicial quadrature-based finite element operators using Bernstein polynomials*, Numer. Math., 121 (2012), pp. 261–279.
[17] R. C. Kirby, M. Knepley, A. Logg, and L. R. Scott, *Optimizing the evaluation of finite element matrices*, SIAM J. Sci. Comput., 27 (2005), pp. 741–758.
[18] R. C. Kirby and A. Logg, *A compiler for variational forms*, ACM Trans. Math. Software, 32 (2006), pp. 417–444.
[19] R. C. Kirby and A. Logg, *Benchmarking domain-specific compiler optimizations for variational forms*, ACM Trans. Math. Software, 35 (2008), 10.
[20] M.-J. Lai and L. L. Schumaker, *Spline Functions on Triangulations*, Encyclopedia Math. Appl. 110, Cambridge University Press, Cambridge, UK, 2007.
[21] A. Logg, K.-A. Mardal, and G. N. Wells, *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, Lect. Notes Comput. Sci. Eng. 84, Springer, Berlin, Heidelberg, 2012.
[22] A. Logg and G. N. Wells, *DOLFIN: Automated finite element computing*, ACM Trans. Math. Software, 37 (2010), 20.
[23] P. Monk, *Finite Element Methods for Maxwell's Equations*, Oxford University Press, New York, 2003.
[24] J.-C. Nédélec, *Mixed finite elements in* $\mathbf{R}^3$, Numer. Math., 35 (1980), pp. 315–341.
[25] J.-C. Nédélec, *A new family of mixed finite elements in* $\mathbf{R}^3$, Numer. Math., 50 (1986), pp. 57–81.
[26] K. B. Ølgaard and G. N. Wells, *Optimizations for quadrature representations of finite element tensors through automated code generation*, ACM Trans. Math. Software, 37 (2010), 8.
[27] T. E. Oliphant, *Guide to NumPy*, Vol. 1, Trelgol Publishing, Spanish Fork, UT, 2006.
[28] S. A. Orszag, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92.
[29] C. E. Powell and D. Silvester, *Optimal preconditioning for Raviart–Thomas mixed formulation of second-order elliptic problems*, SIAM J. Matrix Anal. Appl., 25 (2004), pp. 718–738.
[30] P. A. Raviart and J. M. Thomas, *A mixed finite element method for 2nd order elliptic problems*, in Mathematical Aspects of Finite Element Methods (Proc. Conf., Consiglio Naz. delle Ricerche (C.N.R.), Rome, 1975), Lecture Notes in Math. 606, Springer, Berlin, 1977, pp. 292–315.
[31] M. E. Rognes, R. C. Kirby, and A. Logg, *Efficient assembly of* $H(\mathrm{div})$ *and* $H(\mathrm{curl})$ *conforming finite elements*, SIAM J. Sci. Comput., 31 (2009), pp. 4130–4151.
[32] A. H. Stroud, *Approximate Calculation of Multiple Integrals*, Prentice-Hall Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1971.
[33] I. M. Wilbers, H. P. Langtangen, and Å. Ødegård, *Using Cython to speed up numerical Python programs*, in Proceedings of the 5th National Conference on Computational Mechanics 2009 (MekIT 2009), Tapir Academic Press, Trondheim, Norway, 2009, pp. 495–512.