Numer. Math. (2017) 135:73–95
DOI 10.1007/s00211-016-0795-0

Numerische Mathematik

Fast inversion of the simplicial Bernstein mass matrix

Robert C. Kirby[1]

Received: 15 April 2015 / Revised: 19 November 2015 / Published online: 11 February 2016

Abstract We consider the mass matrix arising from Bernstein polynomials as finite element shape functions. In particular, we give an explicit formula for its eigenvalues and exact characterization of the eigenspaces in terms of the Bernstein representation of orthogonal polynomials. We then derive a fast algorithm for solving linear systems involving the element mass matrix. After establishing these results, we describe the application of Bernstein techniques to the discontinuous Galerkin finite element method for hyperbolic conservation laws, obtaining optimal complexity algorithms. Finally, we present numerical results investigating the accuracy of the mass inversion algorithms and the scaling of total run-time for the function evaluation needed in DG time-stepping.

Keywords Bernstein polynomials · Finite element methods · Fast algorithms

Mathematics Subject Classification 65M60 · 65Y20

1 Introduction

Bernstein polynomials, which are "geometrically decomposed" in the sense of [3] and rotationally symmetric, provide a flexible and general-purpose set of simplicial finite element shape functions. Morever, recent research has demonstrated distinct algorith-

The author acknowledges support from NSF grant CCF-1325480.

✉ Robert C. Kirby
Robert_Kirby@baylor.edu

[1] Department of Mathematics, Baylor University, One Bear Place #97328, Waco, TX 76798-7328, USA

mic advantages over other simplicial shape functions, as many essential elementwise finite element computations can be performed with optimal complexity. In [19], we showed how, with constant coefficients, elementwise mass and stiffness matrices could each be applied to vectors in $\mathcal{O}(n^{d+1})$ operations, where n is the degree of the local basis and d is the spatial dimension. Similar blockwise linear algebraic structure enabled quadrature-based algorithms in [21]. Around the same time, Ainsworth et al. [1] showed that the Duffy transform [10] reveals a tensorial structure in the Bernstein basis itself, leading to sum-factored algorithms for polynomial evaluation and moment computation. Moreover, they provide an algorithm that assembles element matrices with $\mathcal{O}(1)$ work per entry that utilizes their fast moment algorithm together with a very special property of the Bernstein polynomials. Work in [2,20] extends these techniques to $H(\mathrm{div})$ and $H(\mathrm{curl})$.

In this paper, we consider a different problem, that of *mass inversion*. Instead of polynomial evaluation or integration, we consider the question of solving linear systems with the mass matrix, which is the matrix consisting of pairwise inner products of the basis functions. Equivalently, given the moments of a polynomial against the Bernstein basis, how quickly can one recover that polynomial? We rely on the recursive block structure described in [19] to give an $\mathcal{O}(n^{d+1})$ algorithm for solving linear systems with the constant-coefficient mass matrix.

After surveying existing Bernstein finite element techniques in Sect. 2, we then derive our major results in Sect. 3. From there, we turn to the motivating application for our investigation—discontinuous Galerkin methods for hyperbolic conservation laws. We will show how each term in the DG formulation can be handled efficiently. For the element and boundary flux terms, this requires only an adaptation of existing techniques, but our new algorithm is critical lest the mass inversion phase dominate the complexity of the entire process. Finally, we present some numerical results investigating the accuracy of our methods and the overall timing of a DG function evaluation in Sect. 5.

2 Bernstein-basis finite element algorithms

2.1 Notation for Bernstein polynomials

We formulate Bernstein polynomials on the d-simplex using barycentric coordinates and multiindex notation. For a nondegenerate simplex $T \subset \mathbb{R}^d$ with vertices $\{x_i\}_{i=0}^d$, let $\{b_i\}_{i=0}^d$ denote the barycentric coordinates. Each b_i affinely maps $\mathbb{R}^d$ into $\mathbb{R}$ with $b_i(x_j) = \delta_{ij}$ for $0 \le i, j \le d$. It follows that $b_i(x) \ge 0$ for all $x \in T$.

We denote multiindices with Greek letters, although we will begin the indexing with 0 rather than 1. So, $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_d)$ is a tuple of $d+1$ nonnegative integers. We define the *order* of a multiindex α by $|\alpha| \equiv \sum_{i=0}^d \alpha_i$. Factorials and binomial coefficients over multiindices have implied multiplication. That is,

$$\alpha! \equiv \prod_{i=0}^{d} \alpha_i!$$

Without ambiguity of notation, we also define a binomial coefficient with a whole number for the upper argument and multiindex for the lower by

$$\binom{n}{\alpha} = \frac{n!}{\alpha!} = \frac{n!}{\prod_{i=0}^{n} \alpha_i!}.$$

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

Let $\mathbf{b} \equiv (b_0, b_2, \ldots, b_d)$ be a tuple of barycentric coordinates on a simplex. For multiindex α, we define a *barycentric monomial* by

$$\mathbf{b}^\alpha = \prod_{i=0}^{d} b_i^\alpha.$$

We obtain the Bernstein polynomials by scaling these by certain binomial coefficients

$$B_\alpha^n = \frac{n!}{\alpha!}\mathbf{b}^\alpha. \tag{1}$$

For all spatial dimensions and degrees n, the Bernstein polynomials of degree n

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

form a nonnegative partition of unity and a basis for the vector space of polynomials of degree n. They are suitable for assembly in a C^0 fashion or even into smoother splines [24]. For discontinuous Galerkin or other methods that do not require assembly, the geometric decomposition still simplifies handling the boundary terms relative to many other bases.

Crucial to fast algorithms using the Bernstein basis, as originally applied to C^0 elements [1,19], is the sparsity of differentiation. That is, it takes no more than $d + 1$ Bernstein polynomials of degree $n - 1$ to represent the derivative of a Bernstein polynomial of degree n.

For some coordinate direction s, we use the general product rule to write

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

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

$$\frac{\partial B_\alpha^n}{\partial s} = n\sum_{i=0}^{d} B_{\alpha - e_i}^{n-1}\frac{\partial b_i}{\partial s}, \tag{2}$$

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

$$\nabla B_\alpha^n = n \sum_{i=0}^{d} B_{\alpha - e_i}^{n-1} \nabla b_i. \tag{3}$$

Note that each ∇b_i is a fixed vector in $\mathbb{R}^n$ for a given simplex T. In [20], we provide a data structure called a *pattern* for representing gradients as well as exterior calculus basis functions. For implementation details, we refer the reader back to [20].

The *degree elevation* operator will also play a crucial role in our algorithms. This operator expresses a B-form polynomial of degree $n-1$ as a degree n polynomial in B-form. For the orthogonal and hierarchical bases in [18], this operation would be trivial – appending the requisite number of zeros in a vector, while for Lagrange bases it is typically quite dense. While not trivial, degree elevation for Bernstein polynomials is still efficient. Take any Bernstein polynomial and multiply it by $\sum_{i=0}^{d} b_i = 1$ to find

$$\begin{aligned} B_\alpha^{n-1} &= \left(\sum_{i=0}^{d} b_i \right) B_\alpha^{n-1} = \sum_{i=0}^{d} b_i B_\alpha^{n-1} \\ &= \sum_{i=0}^{d} \frac{(n-1)!}{\alpha!} \mathbf{b}^{\alpha + e_i} = \sum_{i=0}^{d} \frac{\alpha_i + 1}{n} \frac{n!}{(\alpha + e_i)!} \mathbf{b}^{\alpha + e_i} \\ &= \sum_{i=0}^{d} \frac{\alpha_i + 1}{n} B_{\alpha + e_i}^n. \end{aligned} \tag{4}$$

We could encode this operation as a $P_n^d \times P_{n-1}^d$ matrix consisting of exactly $d+1$ nonzero entries per column, but it can also be applied with a simple nested loop. At any rate, we denote this linear operator as $E^{d,n}$, where n is the degree of the resulting polynomial. We also denote E^{d,n_1,n_2} the operation that successively raises a polynomial from degree n_1 into n_2. This is just the product of $n_2 - n_1$ (sparse) operators:

$$E^{d,n_1,n_2} = E^{d,n_2} \dots E^{d,n_1+1}. \tag{5}$$

We have that $E^{d,n} = E^{d,n-1,n}$ as a special case.

2.2 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 [21], but the connection to the Duffy transform and decomposition of Bernstein polynomials was quite cleanly presented by Ainsworth et al. in [1].

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

$$\lambda_0 = t_1 \tag{6}$$

and then inductively by

$$\lambda_i = t_{i+1}\left(1 - \sum_{j=0}^{i-1} \lambda_j\right) \tag{7}$$

for $1 \leq i \leq d-1$, and then finally

$$\lambda_d = 1 - \sum_{j=0}^{d-1} \lambda_j. \tag{8}$$

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

$$\mathbf{x}(\mathbf{t}) = \sum_{i=0}^{d} \mathbf{x}_i \lambda_i(\mathbf{t}) \tag{9}$$

maps the unit d-cube onto T.

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

$$\int_T f(\mathbf{x})d\mathbf{x} = \frac{|T|}{d!}\int_0^1 dt_1(1-t_1)^{d-1}\int_0^1 dt_2(1-t_2)^{d-2}\ldots\int_0^1 dt_t f(\mathbf{x}(t)). \tag{10}$$

The *Stroud conical rule* [29] 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 [18] among many other places.

As proven in [1], pulling the Bernstein basis back to $[0,1]^d$ under the Duffy transform reveals a tensor-like structure. It is shown that with $B_i^n(t) = \binom{n}{i}t^i(1-t)^{n-i}$ the one-dimensional Bernstein polynomial, that

$$B_\alpha^n(\mathbf{x}(\mathbf{t})) = B_{\alpha_0}^n(t_1)B_{\alpha_1}^{n-\alpha_0}(t_2)\cdots B_{\alpha_{d-1}}^{n-\sum_{i=0}^{d-2}\alpha_i}(t_d). \tag{11}$$

This is a "ragged" rather than true tensor product, much as the collapsed coordinate simplicial bases [18], but entirely sufficient to enable sum-factored algorithms.

2.3 Basic algorithms

The Stroud conical rule and tensorialization of Bernstein polynomials under the Duffy transformation 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 turns out to be fundamental for optimal assembly and application of Bernstein-basis bilinear forms. First, any polynomial

$u(\mathbf{x}) = \sum_{|\alpha|=n} \mathrm{u}_\alpha B^n_\alpha(\mathbf{x})$ may be evaluated at the Stroud conical points in $\mathcal{O}(n^{d+1})$ operations. In [21], this result is presented as exploiting 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

$$\mu^n_\alpha(f) = \int_T f(\mathbf{x}) B^n_\alpha d\mathbf{x}$$

for all $|\alpha| = n$ via Stroud quadrature in $\mathcal{O}(n^{d+1})$ operations. In the case where f is constant on T, we may also use the algorithm for applying a mass matrix in [19] 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 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:

$$B^{n_1}_\alpha B^{n_2}_\beta = \frac{\binom{\alpha+\beta}{\alpha}}{\binom{n_1+n_2}{n_1}} B^{n_1+n_2}_{\alpha+\beta}, \tag{12}$$

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 $\mathcal{O}(n^{d+1})$ entries. In [20], we show how to adapt these algorithms to short linear combinations of Bernstein polynomials so that stiffness and convection matrices require the same order of complexity as the mass.

3 The Bernstein mass matrix

We begin by defining the rectangular Bernstein mass matrix on a d-simplex T by

$$M^{T,m,n}_{\alpha\beta} = \int_T B^m_\alpha B^n_\beta dx, \tag{13}$$

where $m, n \geq 0$.

By a change of variables, we can write

$$M^{T,m,n} = M^{d,m,n}|T|d!, \tag{14}$$

where $M^{d,m,n}$ is the mass matrix on the unit right simplex S_d in d-space and $|T|$ is the d-dimensional measure of T. When $m = n$, we suppress the third superscript and write $M^{T,m}$ or $M^{d,m}$. We include the more general case of a rectangular matrix because such will appear later in our discussion of the block structure.

This mass matrix has many beautiful properties. Besides the block-recursive structure developed in [19], it is related to the Bernstein-Durrmeyer operator [8,12] of approximation theory. Via this connection, we provide an exact characterization of its eigenvalues and associated eigenspaces in the square case $m = n$. Finally, and most pertinent to the case of discontinuous Galerkin methods, we describe algorithms for solving linear systems involving the mass matrix.

Before proceeding, we recall from [19] that, using formulae for integrals of products of powers of barycentric coordinates, the mass matrix formula is exactly

$$M^{d,m,n}_{\alpha,\beta} = \frac{m!n!\,(\alpha+\beta)!}{(m+n+d)!\alpha!\beta!} \tag{15}$$

3.1 Spectrum

The Bernstein-Durrmeyer operator [8] is defined on L^2 by

$$D_n(f) = \frac{(n+d)!}{n!} \sum_{|\alpha|=n} \left(f, B^n_\alpha\right) B^n_\alpha. \tag{16}$$

This has a structure similar to a discrete Fourier series, although the Bernstein polynomials are not orthogonal. The original Bernstein operator [24] has a structure similar to a Lagrange interpolant, although the basis is not interpolatory.

For $i \geq 1$, we let Q_i denote the space of d-variate polynomials of degree i that are L^2 orthogonal to all polynomials of degree $i-1$ on the simplex. The following result is given in [8], and also referenced in [12] to generate the B-form of simplicial orthogonal polynomials.

Theorem 1 (Derriennic) *For each* $0 \leq i \leq n$, *each*

$$\lambda_{i,n} = \frac{(n+d)!n!}{(n+i+d)!\,(n-i)!}$$

is an eigenvalue of D_n *corresponding to the eigenspace* Q_i.

This gives a sequence of eigenvalues $\lambda_{0,n} > \lambda_{1,n} > \cdots > \lambda_{n,n} > 0$, each corresponding to polynomial eigenfunctions of increasing degree.

Up to scaling, the Bernstein-Durrmeyer operator restricted to polynomials P_n exactly corresponds to the action of the mass matrix. To see this, suppose that $P_n \ni p = \sum_{|\alpha|=n} \mathrm{p}_\alpha B^n_\alpha$. Then

$$\begin{aligned}
\tfrac{n!}{(n+d)!} D_n(p) &= \textstyle\sum_{|\alpha|=n} \left(p, B^n_\alpha\right) B^n_\alpha \\
&= \textstyle\sum_{|\alpha|=n} \left(\sum_{|\beta|=n} \mathrm{p}_\beta B^n_\beta, B^n_\alpha\right) B^n_\alpha \\
&= \textstyle\sum_{|\alpha|=n} \sum_{|\beta|=n} \mathrm{p}_\beta \left(B^n_\beta, B^n_\alpha\right) B^n_\alpha \\
&= \textstyle\sum_{|\alpha|=n} \left(\sum_{|\beta|=n} M^n_{\alpha,\beta} \mathrm{p}_\beta\right) B^n_\alpha
\end{aligned} \tag{17}$$

This shows that the coefficients of the B-form of $\frac{n!}{(n+d)!} D_n(p)$ are just the entries of the Bernstein mass matrix times the coefficients of p. Consequently,

Theorem 2 *For each* $0 \leq i \leq n$,

$$\lambda_{i,n} = \frac{(n+d)!\,(n!)^2}{(n+i+d)!\,(n-i)!}$$

is an eigenvalue of M^n *of multiplicity of* $\binom{d+i-1}{d-1}$, *and the eigenspace is spanned by the B-form of any basis for* Q_i.

This also implies that the Bernstein mass matrices are quite ill-conditioned in the two norm, using the characterization in terms of extremal eigenvalues for SPD matrices.

Corollary 1 *The 2-norm condition number of* $M^{d,n}$ *is*

$$\frac{\lambda_{0,n}}{\lambda_{n,n}} = \frac{(2n+d)!}{(n+d)!n!} \tag{18}$$

However, the spread in eigenvalues does not tell the whole story. We have exactly $n+1$ distinct eigenvalues, independent of the spatial dimension. This shows significant clustering of eigenvalues when $d \geq 1$.

Corollary 2 *In exact arithmetic, unpreconditioned conjugate gradient iteration will solve a linear system of the form* $M^{d,n}x = y$ *in exactly* $n+1$ *iterations, independent of* d.

If the fast matrix-vector algorithms in [1,19] are used to compute the matrix-vector product, this gives a total operation count of $\mathcal{O}(n^{d+2})$. Interestingly, this ties the per-element cost of Cholesky factorization when $d = 2$, but without the startup or storage cost. It even beats a pre-factored matrix when $d \geq 2$. However, in light of the large condition number given by Corollary 1, it is doubtful whether this iteration count can be realized in actual floating point arithmetic. Moreover, it turns out that we can do better and obtain a $\mathcal{O}(n^{d+1})$ algorithm, to which we turn shortly.

Besides complicating the convergence of iterative methods, the high condition number of the mass matrix also suggests a possible additional source of error in finite element element calculations. How badly does roundoff error in inverting the mass matrix perturb the associated member of the finite element space? With Bernstein polynomials, we can answer this with some precision. Suppose that we commit a (forward) error of size ϵ in solving $Mx = y$, computing instead some $\hat{x}$ such that $\|x - \hat{x}\| = \epsilon$ in the vector ∞ norm. Both the actual solution vector x and the erroneous solution $\hat{x}$ can be identified with polynomials u and $\hat{u}$ over some simplex K by taking them as the coefficients of the Bernstein polynomials in a linear combination. Because a polynomial in B-form lies in the convex hull of its control points [24], we also know that u and $\hat{u}$ satisfy

$$\max_{x \in K} |u(x) - \hat{u}(x)| \leq \|x - \hat{x}\|_\infty .$$

Although the mass matrix is ill-conditioned, arithmetic errors in its solution does not lead to disproportionate errors in the finite element functions. Supposing 10-digit numerical linear algebra accuracy, which our empirical results later show is typically achieved, we would only incur an additional maximum pointwise error of 10^{-10} in the associated finite element functions resulting from roundoff in mass inversion.

3.2 Block structure and a fast solution algorithm

Here, we recall several facts proved in [19] related to the block structure of $M^{d,m,n}$, which we will apply now for solving square systems.

We consider partitioning the mass matrix formula (15) by freezing the first entry in α and β. Since there are $m+1$ possible values for for α_0 and $n+1$ for β_0, this partitions $M^{d,m,n}$ into an $(m+1)\times(n+1)$ array, with blocks of varying size. In fact, each block $M^{d,m,n}_{\alpha_0,\beta_0}$ is $P^{d-1}_{m-\alpha_0}\times P^{d-1}_{n-\beta_0}$.

These blocks are themselves, up to scaling, Bernstein mass matrices of lower dimension. In particular, we showed that

$$M^{d,m,n}_{\alpha_0,\beta_0} = \frac{\binom{m}{\alpha_0}\binom{n}{\beta_0}}{\binom{m+n+d+1}{\alpha_0+\beta_0}\,(m+n+d)} M^{d-1,m-\alpha_0,n-\beta_0}. \tag{19}$$

We introduce the $(m+1)\times(n+1)$ array consisting of the scalars multiplying the lower-dimensional mass matrices as

$$\nu^{d,m,n}_{\alpha_0,\beta_0} = \frac{\binom{m}{\alpha_0}\binom{n}{\beta_0}}{\binom{m+n+d+1}{\alpha_0+\beta_0}\,(m+n+d)} \tag{20}$$

so that $M^{d,m,n}$ satisfies the block structure, with superscripts on ν terms dropped for clarity, of

$$M^{d,m,n} = \begin{pmatrix} \nu_{0,0}M^{d-1,m,n} & \nu_{0,1}M^{d-1,m,n-1} & \dots & \nu_{0,n}M^{d-1,m,0} \\ \nu_{1,0}M^{d-1,m-1,n} & \nu_{1,1}M^{d-1,m-1,n-1} & \dots & \nu_{1,n}M^{d-1,m-1,0} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_{n,0}M^{d-1,0,n} & \nu_{n,1}M^{d-1,0,n-1} & \dots & \nu_{n,n}M^{d-1,0,0} \end{pmatrix}. \tag{21}$$

We partition the right-hand side and solution vectors y and x conformally to M, so that the block y_j is of dimension P^{d-1}_{n-j} and corresponds to a polynomial's B-form coefficients with first indices equal to j. We write the linear system in an augmented block matrix as

$$\left(\begin{array}{cccc|c} \nu_{0,0}M^{d-1,n,n} & \nu_{0,1}M^{d-1,n,n-1} & \dots & \nu_{0,n}M^{d-1,n,0} & y_0 \\ \nu_{1,0}M^{d-1,n-1,n} & \nu_{1,1}M^{d-1,n-1,n-1} & \dots & \nu_{1,n}M^{d-1,n-1,0} & y_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \nu_{n,0}M^{d-1,n-1,n} & \nu_{n,1}M^{d-1,n-1,n-1} & \dots & \nu_{n,n}M^{d-1,0,0} & y_n \end{array}\right). \tag{22}$$

From [19], we also know that mass matrices of the same dimension but differing degrees are related via degree elevation operators by

$$M^{d,m-1,n} = \left(E^{d,m}\right)^t M^{d,m,n}. \tag{23}$$

and

$$M^{d,m,n-1} = M^{d,m,n} E^{d,n}. \tag{24}$$

Iteratively, these results give

$$M^{d,m-i,n} = \left(E^{d,m-i,m}\right)^T M^{d,m,n}. \tag{25}$$

for $1 \leq i \leq m$ and

$$M^{d,m,n-j} = M^{d,m,n} E^{d,n-j,n} \tag{26}$$

for $1 \leq j \leq n$. In [19], we used these features to provide a fast algorithm for matrix multiplication, but here we use them to efficiently solve linear systems.

Carrying out blockwise Gaussian elimination in (22), we multiply the first row, labeled with 0 rather than 1, by $\frac{\nu_{1,0}}{\nu_{0,0}} M^{d-1,n-1,n} \left(M^{d-1,n,n}\right)^{-1}$ and subtract from row 1 to introduce a zero block below the diagonal. However, this simplifies, as (23) tells us that

$$\begin{aligned} M^{d-1,n-1,n} \left(M^{d-1,n,n}\right)^{-1} &= \left(E^{d-1,n}\right)^t M^{d-1,n,n} \left(M^{d-1,n,n}\right)^{-1} \\ &= \left(E^{d-1,n}\right)^t. \end{aligned} \tag{27}$$

Because of this, along row 1 for $j \geq 1$, the elimination step gives entries of the form

$$\nu_{1j} M^{d-1,n-1,n-j} - \frac{\nu_{10}\nu_{0j}}{\nu_{00}} \left(E^{d-1,n}\right)^t M^{d-1,n,n-j},$$

but (23) renders this as simply

$$\nu_{1j} M^{d-1,n-1,n-j} - \frac{\nu_{10}\nu_{0j}}{\nu_{00}} M^{d-1,n-1,n-j} = \left(\nu_{1j} - \frac{\nu_{10}\nu_{0j}}{\nu_{00}}\right) M^{d-1,n-1,n-j}. \tag{28}$$

That is, the row obtained by block Gaussian elimination is the same as one would obtain simply by performing a step of Gaussian elimination on the matrix of coefficients $N^{d,n}$ containing the ν values above, as the matrices those coefficients scale do not change under the row operations. Hence, performing elimination on the $(n+1) \times (n+1)$

matrix, independent of the dimension d, forms a critical step in the elimination process. After the block upper triangularization, we arrive at a system of the form

$$\left(\begin{array}{cccc|c} \widetilde{\nu}_{0,0} M^{d-1,n,n} & \widetilde{\nu}_{0,1} M^{d-1,n,n-1} & \dots & \widetilde{\nu}_{0,n} M^{d-1,n,0} & \widetilde{y}_0 \\ 0 & \widetilde{\nu}_{1,1} M^{d-1,n-1,n-1} & \dots & \widetilde{\nu}_{1,n} M^{d-1,n-1,0} & \widetilde{y}_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \widetilde{\nu}_{n,n} M^{d-1,0,0} & \widetilde{y}_n \end{array}\right), \tag{29}$$

where the tildes denote that quantities were updated through elimination. The backward substition proceeds along similar lines, though it requires the solution of linear systems with mass matrices in dimension $d-1$. Multiplying through each block row by $\frac{1}{\widetilde{\nu}_{i,i}} (M^{d-1,n-i})^{-1}$ then gives, using (26)

$$\left(\begin{array}{cccc|c} I & \widetilde{\nu}'_{0,1} E^{d-1,n-1,n} & \dots & \widetilde{\nu}'_{0,n} E^{d-1,0,n} & \widetilde{y}'_0 \\ 0 & I & \dots & \widetilde{\nu}'_{1,n} E^{d-1,0,n-1} & \widetilde{y}'_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I & \widetilde{y}'_n \end{array}\right), \tag{30}$$

where the primes denote quantities updated in the process. We reflect this in the updated N matrix by scaling each row by its diagonal entry as we proceed. At this point, the last block of the solution is revealed, and can be successively elevated, scaled, and subtracted from the right-hand side to eliminate it from previous blocks. This reveals the next-to last block, and so-on. We summarize this discussion in Algorithm 1.

Since we will need to solve many linear systems with the same element mass matrix, it makes sense to extend our elimination algorithm into a reusable factorization. We will derive a blockwise LDL^T factorization of the element matrix, very much along the lines of the standard factorizatin [28].

Let $N^{d,n}$ be the matrix of coefficients given in (20). Suppose that we have its LDL^T factorization

$$N^{d,n} = L_N^{d,n} D_N^{d,n} \left(L_N^{d,n}\right)^t, \tag{31}$$

with ℓ_{ij} and d_{ii} the entries of $L_N^{d,n}$ and $D_N^{d,n}$, respectively. We also define $U_N^{1,n} = D_N^{1,n} \left(L_N^{1,n}\right)^t$ with $u_{ij} = d_{ii}\ell_{ji}$

Then, we can use the block matrix

$$\widetilde{L}^0 = \begin{pmatrix} I & 0 \dots 0 \\ -\ell_{10} \left(E^{d-1,n-1,n}\right)^T & I \dots 0 \\ -\ell_{20} \left(E^{d-1,n-2,n}\right)^T & 0 \dots 0 \\ \vdots & \vdots \ddots \vdots \\ -\ell_{n0} \left(E^{d-1,0,n}\right)^T & 0 \dots I \end{pmatrix}$$

Algorithm 1 Block-wise Gaussian elimination for solving $M^{d,n}x = y$

Require: Input vector y
Ensure: On output, y is overwritten with $(M^{d,n})^{-1}y$
Initialize coefficient matrix $N_{a,b} := \frac{\binom{n}{a}\binom{n}{b}}{\binom{2n+d+1}{a+b}(2n+d)}$
for $a := 0$ **to** n **do** {Forward elimination}
  $z \leftarrow y_a$
  for $b := a+1$ **to** n **do**
    $z \leftarrow (E^{n-1,d-b+1})^T z$
    $y_b \leftarrow y_b - \frac{N_{b,a}}{N_{a,a}} z$
    for $c := a$ **to** n **do** {Elimination on N}
      $N_{b,c} \leftarrow N_{b,c} - \frac{N_{b,a}N_{a,c}}{N_{a,a}}$
    end for
  end for
end for
for $a := 0$ **to** n **do** {Lower-dimensional inversion}
  $y_a \leftarrow \frac{1}{N_{a,a}} \left(M^{d-1,n-a,n-a}\right)^{-1} y_a$
  for $b := a$ **to** n **do**
    $N_{b,a} \leftarrow \frac{N_{b,a}}{N_{a,a}}$
  end for
end for
for $a := n$ **to** 0 **do** {Backward elimination}
  $z \leftarrow y_a$
  for $b := a-1$ **to** 0 **do**
    $z = E^{d-1,n-b} z$
    $y_b \leftarrow y_b - N_{b,a} z$
  end for
end for

to act on $M^{d,n}$ to produce zeros below the diagonal in the first block of columns. Similarly, we act on $\widetilde{L}^0 M^{d,n}$ with

$$\widetilde{L}^1 = \begin{pmatrix} I & 0 & \dots 0 \\ 0 & I & \dots 0 \\ 0 & -\ell_{21}\left(E^{d-1,n-2,n-1}\right)^T & \dots 0 \\ \vdots & \vdots & \ddots \vdots \\ 0 & -\ell_{n1}\left(E^{d-1,0,n-1}\right)^T & \dots I \end{pmatrix}$$

to introduce zeros below the diagonal in the second block of columns. Indeed, we have a sequence of block matrices $\widetilde{E}^k$ for $0 \le k < n$ such that $\widetilde{L}^k_{ij}$ is $P^{d-1}_{n-i} \times P^{d-1}_{n-j}$ with

$$\widetilde{L}^k_{ij} = \begin{cases} I & \text{for } i = j \\ 0 & \text{for } i \neq j \text{ and } j \neq k \\ 0 & \text{for } i < j \text{ and } j = k \\ -\ell_{ij}\left(E^{d-1,n-i,n-j}\right)^T & \text{for } i > j \text{ and } j = k \end{cases}$$

Then, in fact, we have that

$$\widetilde{L}^{n-1}\widetilde{L}^{n-2}\dots\widetilde{L}^{0}M^{d,n} = \begin{pmatrix} u_{00}M^{d-1,n,n} & u_{01}M^{d-1,n,n-1} & \dots & u_{0n}M^{d-1,n,0} \\ 0 & u_{11}M^{d-1,n-1,n-1} & \dots & u_{1n}M^{d-1,n-1,0} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn}M^{d-1,0,0} \end{pmatrix}$$

Much as with elementary row matrices for classic LU factorization, we can invert each of these $\widetilde{L}^k$ matrices simply by flipping the sign of the multiplier, so that

$$\left(\widetilde{L}^k\right)^{-1}_{ij} = \begin{cases} I & \text{for } \; i = j \\ 0 & \text{for } \; i \neq j \text{ and } j \neq k \\ 0 & \text{for } \; i < j \text{ and } j = k \\ \ell_{ij}\left(E^{d-1,n-i,n-j}\right)^T & \text{for } \; i > j \text{ and } j = k \end{cases}.$$

Then, we define $L^{d,n}$ to be the inverse of these products

$$L^{d,n} = \left(\widetilde{L}^{n-1}\widetilde{L}^{n-2}\dots\widetilde{L}^{0}\right)^{-1} = \left(\widetilde{L}^{0}\right)^{-1}\left(\widetilde{L}^{1}\right)^{-1}\dots\left(\widetilde{L}^{n-1}\right)^{-1} \tag{32}$$

so that $\left(L^{d,n}\right)^{-1}M^{d,n} \equiv U^{d,n}$ is block upper triangular. Like standard factorization, we can also multiply the elimination matrices together so that

$$\left(L^{d,n}\right)^{-1} = \begin{pmatrix} I & 0 & \dots & 0 \\ -\ell_{10}\left(E^{d-1,n-1,n}\right)^t & I & \dots & 0 \\ -\ell_{20}\left(E^{d-1,n-2,n}\right)^t & -\ell_{21}\left(E^{d-1,n-2,n-1}\right)^t & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\ell_{n0}\left(E^{d-1,0,n}\right)^t & -\ell_{n1}\left(E^{d-1,0,n-1}\right)^t & \dots & I \end{pmatrix}. \tag{33}$$

Moreover, we can turn the block upper triangular matrix into a block diagonal one times the transpose of $L^{d,n}$ giving a kind of block LDL^T factorization. We factor out the pivot blocks from each row, using (27) so that

$$U^{d,n} = \begin{pmatrix} d_{00}M^{d-1,n,n} & 0 & \dots & 0 \\ 0 & d_{11}M^{d-1,n-1,n-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nn}M^{d-1,0,0} \end{pmatrix} \times \begin{pmatrix} I & \ell_{10}E^{d-1,n-1,n} & \dots & \ell_{n0}E^{d-1,0,n} \\ 0 & I & \dots & \ell_{n1}E^{d-1,0,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I \end{pmatrix}.$$

The factor on the right is just $\left(L^{d,n}\right)^T$.

We introduce the block-diagonal matrix $\Delta^{d,n}$ by

$$\Delta_{ii} = d_{ii} M^{d-1,n-i}. \tag{34}$$

Our discussion has established:

Theorem 3 *The Bernstein mass matrix $M^{d,n}$ admits the block factorization*

$$M^{d,n} = L^{d,n} \Delta^{d,n} \left(L^{d,n} \right)^T. \tag{35}$$

We can apply the decomposition inductively down spatial dimension, so that each of the blocks in $\Delta^{d,n}$ can be also factored according to Theorem 3. This fully expresses any mass matrix as a diagonal matrix sandwiched in between sequences of sparse unit triangular matrices.

So, computing the LDL^T factorization of $M^{d,n}$ requires computing the LDL^T factorization of the one-dimensional coefficient matrix $N^{d,n}$. Supposing we use standard direct method such as Cholesky factorization to solve the one-dimensional mass matrices in the base case, we will have a start-up cost of factoring $n+1$ matrices of size no larger than $n+1$. With Cholesky, this is a $\mathcal{O}(n^4)$ process, although since the one-dimensional matrices factor into Hankel matrices pre- and post-multiplied by diagonal matrices, one could use Levinson's or Bareiss' algorithm [4,25] to obtain a merely $\mathcal{O}(n^3)$ startup phase.

Algorithm 2 Mass inversion via block-recursive LDL^T factorization for $d \geq 2$

Require: $N^{d,n}$ factored as $N^{d,n} = LDL^T$
Require: Input vector y
Ensure: On output, $x = (M^{d,n})^{-1} y$
  Initialize vector $x \leftarrow 0$
  for $a := 0$ **to** n **do** {Apply $(L^{d,n})^{-1}$ to y, store in x}
    $z \leftarrow y_a$
    for $b := a+1$ **to** n **do**
      $z \leftarrow \left(E^{d-1,n-b+1} \right)^t$
      $x_b \leftarrow x_b - L_{b,a} z$
    end for
  end for
  for $a := 0$ **to** n **do** {Overwrite x with $(\Delta^{d,n})^{-1} x$}
    $x_a \leftarrow \frac{1}{D_{a,a}} M^{d,n-a} x_a$
  end for
  for $a := n$ **to** 0 **do** {Overwrite x with $(L^{n,d})^{-T} x$}
    $z \leftarrow x_a$
    for $b := a-1$ **to** 0 **do**
      $z \leftarrow E^{d-1,n-b} z$
      $x_b \leftarrow x_b - L_{b,a} z$
    end for
  end for

Now, we also consider the cost of solving a linear system using the block factorization, pseudocode for which is presented in Algorithm 2. In two dimensions, one must

apply the inverse of $L^{2,n}$, followed by the inverse of $\Delta^{2,n}$, accomplished by triangular solves using pre-factored one-dimensional mass matrices, and the inverse of $(L^{2,n})^T$. In fact, the action of applying $(L^{2,n})^{-1}$ requires exactly the same process as described above for block Gaussian elimination, except the arithmetic on the v values is handled in preprocessing. That is, for each block y_j, we will need to compute $\ell_{ij}(E^{1,j-i,j})^T y_j$ for $1 \le i \le n-j-1$ and accumulate scalings of these vectors into corresponding blocks of the result. Since these elevations are needed for each i, it is helpful to reuse these results. Applying $(L^{2,n})^{-1}$ then requires applying $E^{1,i-j}$ for all valid i and j, together with all of the axpy operations. Since the one-dimensional elevation into degree i has $2(i+1)$ nonzeros in it, the required elevations required cost

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1} 2(j+1) = \frac{n(n^2+3n-4)}{3}, \tag{36}$$

operations, which is $\mathcal{O}(n^3)$, and we also have a comparable number of operations for the axpy-like operations to accumulate the result. A similar discussion shows that applying $(L^{2,n})^{-T}$ requires the same number of operations. Between these stages, one must invert the lower-dimensional mass matrices using the pre-computed Cholesky factorizations and perform the scalings to apply Δ^{-1}. Since a pair of $m \times m$ triangular solves costs $m(m+1)$ operations, the total cost of the one-dimensional mass inversions is

$$\sum_{i=0}^{n}(i+1)(i+2) = \frac{(n+1)(n+2)(n+3)}{3},$$

together with the lower-order term for scalings

$$\sum_{i=0}^{n} P_i^1 = \sum_{i=0}^{n}(i+1) = \frac{(n+1)(n+2)}{2}.$$

So, the whole three-stage process is $\mathcal{O}(n^3)$ per element.

In dimension $d > 2$, we may proceed inductively in space dimension to show that Algorithm 2 requires, after start-up, $\mathcal{O}(n^{d+1})$ operations. The application of Δ^{-1} will always require $n+1$ inversions of $(d-1)$-dimensional mass matrices , each of which costs $\mathcal{O}(n^d)$ operations by the induction hypothesis. Inverting $\Delta^{d,n}$ onto a vector will cost $\mathcal{O}(n^{d+1})$ operations for all n and d. To see that a similar complexity holds for applying the inverses of $L^{d,n}$ and its transpose, one can simply replace the summand in (36) with $2P_j^{d-1}$ and execute the sum. To conclude,

Theorem 4 *Algorithm 2 applies the inverse of $M^{d,n}$ to an arbitrary vector in $\mathcal{O}(n^{d+1})$ operations.*

4 Application to discontinuous Galerkin methods

Consider an equation of the form

$$q_t + \nabla \cdot F(q) = 0, \tag{37}$$

posed on a domain $\Omega \times [0, T) \subset \mathbb{R}^d \times \mathbb{R}$, together with suitable initial and boundary conditions. As a particular example, we consider the linear acoustic model

$$\begin{aligned} p_t + \nabla \cdot u &= 0, \\ u_t + \nabla p &= 0, \end{aligned} \tag{38}$$

Here, $q = [u, p]^T$ where the pressure variable p is a scalar-valued function on $\Omega \times [0, T]$ and the velocity u maps the same space-time domain into $\mathbb{R}^d$.

To obtain a discontinuous Galerkin (hence, DG) method [7], we let $\mathcal{T}_h$ be a triangulation of Ω in the sense of [6] into affine simplices. For curved-sided elements, we could adapt the techniques of [32] to incorporate the Jacobian into our local basis functions to recover the reference mass matrix on each cell at the expense of having variable coefficients in other operators, but this does not affect the overall order of complexity. We let $\mathcal{E}_h$ denote the set of all edges in the triangulation.

For $T \in \mathcal{T}_h$, let $P_n(T)$ be the space of polynomials of degree no greater than n on T. This is a vector space of dimension $P_n^d \equiv \binom{n+d}{n}$. We define the global finite element space

$$V_h = \{ f : \Omega \to \mathbb{R} : f|_T \in P_n(T),\ T \in \mathcal{T}_h \}, \tag{39}$$

with no continuity enforced between cells. Let $(\cdot, \cdot)_T$ denote the standard L^2 inner product over $T \in \mathcal{T}_h$ and $\langle \cdot, \cdot \rangle_\gamma$ the L_2 inner product over an edge $\gamma \in \mathcal{E}_h$.

After multiplying (37) by a test function and integating by parts elementwise, a DG method seeks u_h in V_h such that

$$\sum_{T \in \mathcal{T}_h} \left[\left(u_{h,t}, v_h \right)_T - (F(u_h), \nabla v_h)_T \right] + \sum_{\gamma \in \mathcal{E}_h} \langle \hat{F} \cdot n, v_h \rangle = 0 \tag{40}$$

for all $v_h \in V_h$.

Fully specifying the DG method requires defining a numerical flux function $\hat{F}$ on each γ. On internal edges, it takes values from either side of the edge and produces a suitable approximation to the flux F. Many Riemann solvers from the finite volume literature have been adapted for DG methods [7,11,30]. The particular choice of numerical flux does not matter for our purposes. On external edges, we choose $\hat{F}$ to appropriately enforce boundary conditions.

This discretization gives rise to a system of ordinary differential equations

$$\mathrm{M}\mathrm{u}_t + \mathrm{F}(\mathrm{u}) = 0, \tag{41}$$

where M is the block-diagonal mass matrix and F(u) includes the cell and boundary flux terms. Because of the hyperbolic nature of the system, explicit methods are frequently preferred. A forward Euler method, for example, gives

$$\mathrm{u}^{n+1} = \mathrm{u}^n - \Delta t \mathrm{M}^{-1} \mathrm{F}(\mathrm{u}^n) \equiv \mathrm{u}^n - \Delta t L(\mathrm{u}^n), \tag{42}$$

which requires the application of M at each time step. The SSP methods [13,27] give stable higher-order in time methods. For example, the well-known third order scheme is

$$\begin{aligned} \mathrm{u}^{n,1} &= \mathrm{u}^n + \Delta t L(\mathrm{u}^n), \\ \mathrm{u}^{n,2} &= \tfrac{3}{4}\mathrm{u}^n + \tfrac{1}{4}\mathrm{u}^{n,1} + \tfrac{1}{4}\Delta t L(\mathrm{u}^{n,1}), \\ \mathrm{u}^{n+1} &= \tfrac{1}{3}\mathrm{u}^n + \tfrac{2}{3}\mathrm{u}^{n,2} + \tfrac{2}{3}\Delta t L(\mathrm{u}^{n,2}). \end{aligned} \tag{43}$$

As part of each explicit time stepping stage, we must evaluate F(u) and hence $\mathrm{M}^{-1}\mathrm{F}(\mathrm{u})$. Evaluating F(u) requires handling the two flux terms in (40). To handle

$$(F(u_h), \nabla v_h)_T \,,$$

we simply evaluate u_h at the Stroud points on T, which requires $\mathcal{O}(n^{d+1})$ operations. Then, evaluating F at each of these points is purely pointwise and so requires but $\mathcal{O}(n^d)$. Finally, the moments against gradients of Bernstein polynomials also requires $\mathcal{O}(n^{d+1})$ operations. This term, then, is readily handled by existing Bernstein polynomial techniques.

Second, we must address, on each interface $\gamma \in \mathcal{E}$,

$$\langle \hat{F} \cdot n, v_h \rangle_\gamma .$$

The numerical flux $\hat{F} \cdot n$ requires the values of u_h on each side of the interface and is evaluated pointwise at each facet quadrature point. Because of the Bernstein polynomials' geometric decomposition, only P_n^{d-1} basis functions are nonzero on that facet, and their traces are in fact exactly the Bernstein polynomials on the facet. So we have to evaluate two polynomials (the traces from each side) of degree n in $d-1$ variables at the facet Stroud points. This requires $\mathcal{O}(n^d)$ operations. The numerical flux is computed pointwise at the $\mathcal{O}(n^{d-1})$ points, and then the moment integration is performed on facets for an overall cost of $\mathcal{O}(n^d)$ for the facet flux term. In fact, the geometric decomposition makes this term much easier to handle optimally with Bernstein polynomials than collapsed-coordinate bases, although though specially adapted Radau-like quadrature rules, the boundary sums may be lifted into the volumetric integration [31].

The mass matrix presents no problem when one employs orthogonal collapsed-coordinate bases [9,18] since it is diagonal. Obtaining full optimality with such bases then requires special quadrature that reflects the tensorial nature of the basis under the Duffy transform or collapsed-coordinate mapping from the d-simplex to the d-cube and also includes appropriate points to incorporate contributions from both volume and boundary flux terms.

Without the fast algorithm described above, however the dense element mass inversion would dominate the overall complexity of the DG method with Bernstein polynomials. With $\mathcal{O}(n^d)$ rows and columns, a standard matrix Cholesky decomposition requires $\mathcal{O}(n^{3d})$ operations as a startup cost, followed by a pair of triangular solves

on each solve at $\mathcal{O}(n^{2d})$ each. For $d > 1$, this dominates the cost at high degree, above, although an optimized Cholesky routine might very well win at moderate orders.

As an alternative to low-complexity algorithms, one could employ more more convential techniques, but phrased in terms of dense matrix multiplication. Hesthaven and Warburton [14,15] use such dense linear algebra in conjunction with Lagrange polynomials. Highly-tuned matrix multiplication can make this approach competitive or even superior at practical polynomial orders. Additional extensions of this idea include the so-called "strong DG" forms and also a pre-elimination of the elementwise mass matrix giving rise to a simple ODE system. With care, this approach can give very high performance on both CPU and GPU systems [22]. We may view our approach, using Bernstein polynomials, as sharing certain important features of both collapsed-coordinate and Lagrange bases. Like collapsed-coordinate methods, we seek to use specialized structure to optimize algorithmic complexity. Like Lagrange polynomials, we seek to do this using a relatively discretization-neutral basis.

DG methods yield reasonable solutions to acoustic or Maxwell's equations without slope limiters, although most nonlinear problems will require them to suppress oscillations. Even linear transport can require limiting when a discrete maximum principle is required. Limiting high-order polynomials on simplicial domains remains quite a challenge. It may be possible to utilize properties of the Bernstein polynomials to design new limiters or conveniently implement existing ones. For example, the convex hull property (i.e. that polynomials in the Bernstein basis lie in the convex hull of their control points) gives sufficient conditions for enforcing extremal bounds. We will not offer further contributions in this direction, but refer the reader to other works on higher order limiting such as [16,33,34].

5 Numerical results

5.1 Mass inversion

Because of Corollary 1, we must pay special attention to the accuracy with which linear systems involving the mass matrix are computed. We began with Cholesky decomposition as a baseline. For degrees one through twenty in one, two, and three space dimension, we explicitly formed the reference mass matrix in Python and used the `scipy` [17] interface LAPACK to form the Cholesky decomposition. Then, we chose several random vectors to be sample solutions and formed the right-hand side by direct matrix-vector multiplication. In Fig. 1, we plot the relative accuracy of a function of degree in each space dimension. Although we observe expontial growth in the error (fully expected in light of Corollary 1), we see that we still obtain at least ten digits of relative accuracy up to degree ten.

Second, we also attempt to solve the linear system using conjugate gradients. We again used systems with random solution, and both letting CG run to a relative residual tolerance of 10^{-12} and also stopping after $n + 1$ iterations in light of Corollary 2. We display the results of a fixed tolerance in Fig. 2. Figure 2a, shows the actual accuracy obtained for each polynomial degree and Fig. 2b gives the actual iteration count required. Like Cholesky factorization, this approach gives nearly ten-digit accuracy

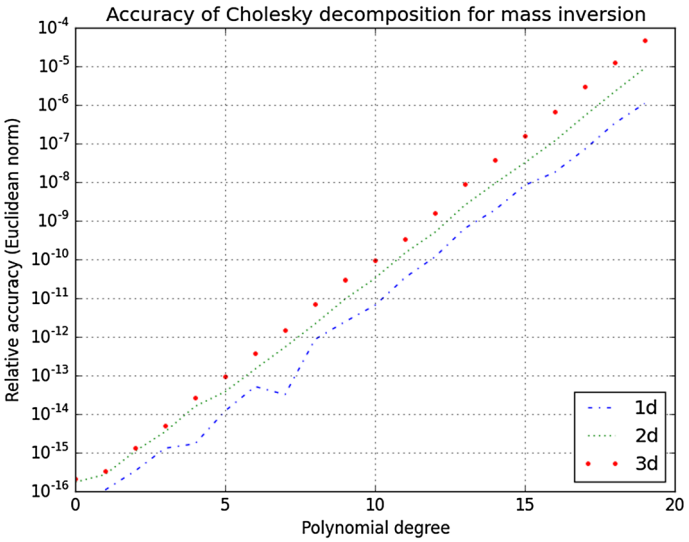

Fig. 1 Relative accuracy of solving linear systems with mass matrices of various degrees using Cholesky decomposition

up to degree ten polynomials. On the other hand, Fig. 3 shows that accuracy degrades markedly when only $n + 1$ iterations are used.

Finally, our block algorithm gives accuracy comparable to that of Cholesky factorization. Our two-dimensional implementation of Algorithm 2 uses Cholesky factorizations of the one-dimensional mass matrices. Rather than full recursion, our three-dimensional implementation uses Cholesky factorization of the two-dimensional matrices. At any rate, Fig. 4 shows, when compared to Fig. 1, that we lose very little additional accuracy over Cholesky factorization. Whether replacing the one-dimensional solver with a specialized method for totally positive matrices [23] would also give high accuracy for the higher-dimensional problems will be the subject of future investigation.

5.2 Timing for first-order acoustics

We fixed a 32×32 square mesh subdivided into right triangles and computed the time to perform the DG function evaluation (including mass matrix inversion) at various polynomial degrees. We used the mesh from DOLFIN [26] and wrote the Bernstein polynomial algorithms in Cython [5]. With an $\mathcal{O}(n^3)$ complexity for two-dimensional problems, we expect a doubling of the polynomial degree to produce an eightfold increase in run-time. In Fig. 5, though, we see even better results. In fact, a least-squares fit of the log-log data in this table from degrees five to fifteen gives a very near fit with a slope of less than two (about 1.7) rather than three. Since small calculations tend to run at lower flop rates, it is possible that we are far from the asymptotical regime predicted by our operation counts.

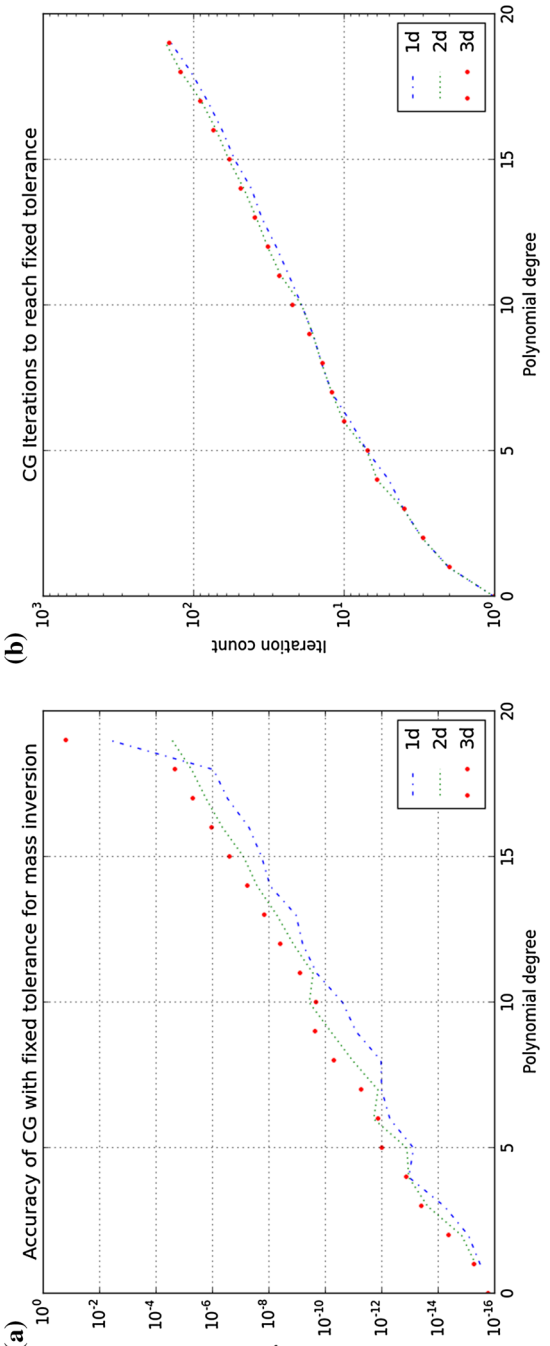

Fig. 2 Accuracy obtained solving mass matrix system using conjugate gradient iteration in one, two, and three space dimensions

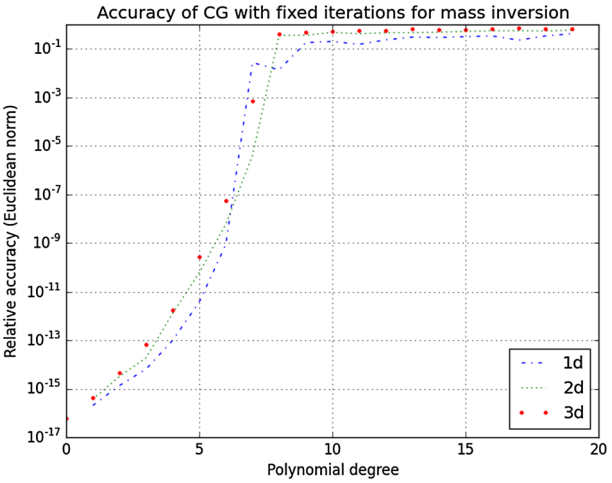

Fig. 3 Relative accuracy of solving $M^{d,n}x = y$ using exactly $n + 1$ CG iterations

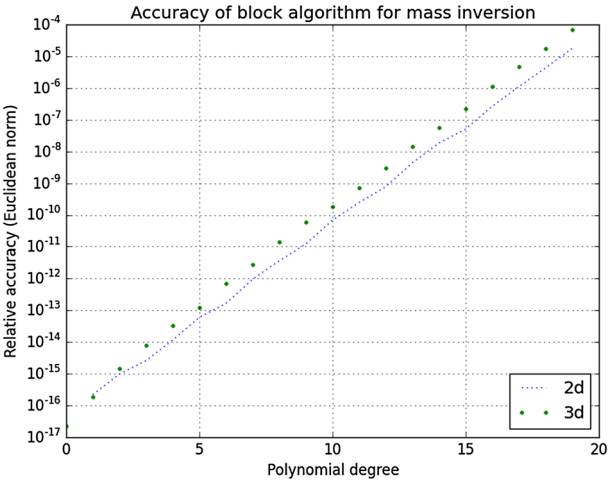

Fig. 4 Relative accuracy of solving linear systems with mass matrices of various degrees using one level of the block algorithm with Cholesky factorization for lower-dimensional matrices

6 Conclusions and future work

Bernstein polynomials admit optimal-complexity algorithms for discontinuous Galerkin methods for conservation laws. The dense element mass matrices might, at first blush, seem to prevent this, but their dimensionally recursive block structure and other interesting properties, lead to an efficient blockwise factorzation. Despite the large condition numbers, our current algorithms seem sufficient to deliver reasonable accuracy at moderate polynomial orders.

On the other hand, these results still leave much room for future investigation. First, it makes sense to explore the possibilities of slope limiting in the Bernstein basis.

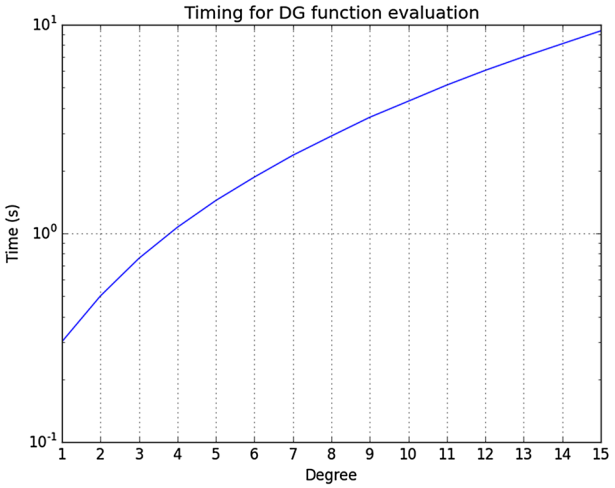

Fig. 5 Timing of a DG function evaluation for various polynomial degrees on a 32×32 mesh

Second, while our mass inversion algorithm is sufficient for moderate order, it may be possible to construct a different algorithm that maintains the low complexity while giving higher relative accuracy, enabling very high approximation orders. Perhaps such algorithms will either utilize the techniques in [23] internally, or else extend them somehow. Finally, our new algorithm, while of optimal compexity, is quite intricate to implement and still is not well-tuned for high performance. Finding ways to make these algorithms more performant will have important practical benefits.

References

1. Ainsworth, M., Andriamaro, G., Davydov, O.: Bernstein-Bézier finite elements of arbitrary order and optimal assembly procedures. SIAM J. Sci. Comp. **33**(6), 3087–3109 (2011)
2. Ainsworth, M., Andriamaro, G., Davydov, O.: A Bernstein-Bézier basis for arbitrary order Raviart-Thomas finite elements. Construct. Approx. **41**(1), 1–22 (2015)
3. Arnold, D.N., Falk, R.S., Winther, R.: Geometric decompositions and local bases for spaces of finite element differential forms. Comp. Methods Appl. Mechan. Eng. **198**(21), 1660–1672 (2009)
4. Bareiss, E.H.: Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices. Numerische Mathematik **13**(5), 404–424 (1969)
5. Behnel, S., Bradshaw, R., Ewing, G.: Cython: C-extensions for Python (2008)
6. Brenner, S.C., Scott, L.R.: The mathematical theory of finite element methods, Texts in Applied Mathematics, vol. 15, third edn. Springer, New York (2008)
7. Cockburn, B., Shu, C.W.: The Runge-Kutta local projection P^1-discontinuous Galerkin finite element method for scalar conservation laws. RAIRO Modél. Math. Anal. Numér. **25**(3), 337–361 (1991)
8. Derriennic, M.M.: On multivariate approximation by Bernstein-type polynomials. J. Approx. Theory **45**(2), 155–166 (1985)
9. Dubiner, M.: Spectral methods on triangles and other domains. J. Sci. Comp. **6**(4), 345–390 (1991)
10. Duffy, M.G.: Quadrature over a pyramid or cube of integrands with a singularity at a vertex. SIAM J. Num. Anal. **19**(6), 1260–1262 (1982)
11. Dumbser, M., Balsara, D.S., Toro, E.F., Munz, C.D.: A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes. J. Comput. Phys. **227**(18), 8209–8253 (2008)

12. Farouki, R.T., Goodman, T.N.T., Sauer, T.: Construction of orthogonal bases for polynomials in Bernstein form on triangular and simplex domains. Comp. Aided Geom. Design **20**(4), 209–230 (2003)
13. Gottlieb, S., Shu, C.W., Tadmor, E.: Strong stability-preserving high-order time discretization methods. SIAM Rev. **43**(1), 89–112 (2001)
14. Hesthaven, J.S., Warburton, T.: Nodal high-order methods on unstructured grids: I. time-domain solution of Maxwell's equations. J. Comput. Phys. **181**(1), 186–221 (2002)
15. Hesthaven, J.S., Warburton, T.: Nodal discontinuous Galerkin methods: algorithms, analysis, and applications, vol. 54. Springer (2007)
16. Hoteit, H., Ackerer, P., Mosé, R., Erhel, J., Philippe, B.: New two-dimensional slope limiters for discontinuous Galerkin methods on arbitrary meshes. Int. J. Num. Methods Eng. **61**(14), 2566–2593 (2004)
17. Jones, E., Oliphant, T., Peterson, P.: SciPy: Open source scientific tools for Python. http://www.scipy.org/ (2001)
18. Karniadakis, G.E., Sherwin, S.J.: Spectral/*hp* element methods for computational fluid dynamics, 2nd edn. Numerical Mathematics and Scientific Computation. Oxford University Press, New York (2005)
19. Kirby, R.C.: Fast simplicial finite element algorithms using Bernstein polynomials. Numerische Mathematik **117**(4), 631–652 (2011)
20. Kirby, R.C.: Low-complexity finite element algorithms for the de Rham complex on simplices. SIAM J. Sci. Comp. **36**(2), A846–A868 (2014)
21. Kirby, R.C., Kieu, T.T.: Fast simplicial quadrature-based finite element operators using Bernstein polynomials. Numerische Mathematik **121**(2), 261–279 (2012)
22. Klöckner, A., Warburton, T., Bridge, J., Hesthaven, J.S.: Nodal discontinuous Galerkin methods on graphics processors. J. Comput. Phys. **228**(21), 7863–7882 (2009)
23. Koev, P.: Accurate computations with totally nonnegative matrices. SIAM J. Matrix Anal. Appl. **29**(3), 731–751 (2007)
24. Lai, M.J., Schumaker, L.L.: Spline functions on triangulations, encyclopedia of mathematics and its applications, vol. 110. Cambridge University Press, Cambridge (2007)
25. Levinson, N.: The Wiener RMS error criterion in filter design and prediction. J. Math. Phys. **25**, 261–278 (1947)
26. Logg, A., Wells, G.N.: DOLFIN: automated finite element computing. ACM Trans. Math. Softw. 37(2): 20–28 (2010). doi:10.1145/1731022.1731030
27. Shu, C.W.: Total-variation-diminishing time discretizations. SIAM J. Sci. Stat. Comp. **9**(6), 1073–1084 (1988)
28. Strang, G.: Linear Algebra and Its Applications. Thomson, Brooks/Cole, Belmont (2006)
29. Stroud, A.H.: Approximate calculation of multiple integrals. Prentice-Hall Inc., Englewood Cliffs, N.J. Prentice-Hall Series in Automatic Computation (1971)
30. Toro, E.F.: Riemann solvers and numerical methods for fluid dynamics, vol. 16. Springer (1999)
31. Warburton, T.: private communication
32. Warburton, T.: A low-storage curvilinear discontinuous Galerkin method for wave problems. SIAM J. Sci. Comp. **35**(4), A1987–A2012 (2013)
33. Zhu, J., Qiu, J., Shu, C.W., Dumbser, M.: Runge-Kutta discontinuous Galerkin method using WENO limiters II: unstructured meshes. J. Comput. Phys. **227**(9), 4330–4353 (2008)
34. Zhu, J., Zhong, X., Shu, C.W., Qiu, J.: Runge-Kutta discontinuous Galerkin method using a new type of WENO limiters on unstructured meshes. J. Comput. Phys. **248**, 200–220 (2013)