# An $\mathcal{O}(p^3)$ *hp*-version FEM in Two Dimensions: Preconditioning and Post-Processing

Mark Ainsworth[a], Shuai Jiang[a,1], Manuel A. Sanchéz[b]

[a] *Division of Applied Mathematics, Brown University*
[b] *Instituto de Ingeniería, Matemática y Computacional, Pontificia Universidad Católica de Chile*

**Abstract**

The Bernstein polynomials have been known for over a century and are widely used in the spline literature, computer aided geometric design, and computer graphics. However, the realisation that the Bernstein basis has favourable properties allowing the efficient implementation of high order methods for the approximation of partial differential equations is a relatively recent development. For instance, it is known [2] that the Bernstein basis can be exploited to compute all of the entries in the load vector in $O(p^3)$ operations even in the case of nonlinear problems on curvilinear elements for a degree $p$ approximation. Moreover, the element matrices can be assembled in $O(1)$ operations per entry. We show that properties of the Bernstein polynomials can also be exploited to obtain $O(p^3)$ complexity procedures for *all* of the main components needed to implement a high order finite element code including: computation of the residuals needed for an iterative solution method; evaluating the action of a preconditioner for the global mass matrices; and, visualization and post-processing of the resulting finite element approximations.

The construction of a preconditioner for the mass matrix whose condition number does not degenerate with the order $p$, at a cost of $O(p^3)$ operations, is one of the main contributions of the present work. The preconditioner is based

*Email addresses:* mark_ainsworth@brown.edu (Mark Ainsworth), shuai_jiang@brown.edu (Shuai Jiang ), manuel.sanchez@ing.puc.cl (Manuel A. Sanchéz)

[1] This work was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

on an abstract Additive Schwarz Method recently developed by the authors. The preconditioner can be implemented at a cost of $O(p^3)$ operations by exploiting properties of the Bernstein polynomials. In particular, we present an algorithm which allows one to invert the interior block of the element mass matrix in $O(p^3)$ operations. Numerical examples are provided to illustrate the applicability of the Bernstein basis to challenging non-linear reaction-diffusion problems, non-linear wave propagation of solitons and to robust approximation of problems exhibiting boundary layers.

*Keywords:* preconditioning mass matrix, high order finite elements, post-processing

*2010 MSC:* 65N30, 65F08, 68U05

---

## 1. Introduction

The theoretical foundations of high order finite element methods (*hp*-FEM) were the topic of intense research in the 1980s-90s, and are now relatively well-established. For instance, *hp*-FEM has been proven to exhibit exponential convergence for many classes of problem, such as PDEs with piecewise analytic data [8], boundary-layer problems [34, 42] and even ones with singularities [14, 42]. Yet there remain practical issues that seem to be inhibiting the more widespread adoption of *hp*-FEM in both industry and academia.

The root cause of many of these issues of *hp*-FEM can often be traced to the selection of an appropriate basis for the implementation. Early endeavors into the construction of high order bases such as Lagrangian and Peano bases quickly fell out of favor due to the condition numbers of the resulting mass and stiffness matrices [45]. Although the current bases of choice are the hierarchical or Dubiner bases [14, 17, 28], recently attention has been drawn to favorable properties of the Bernstein polynomials [2, 30]. The Bernstein polynomials [20] are widely used in the spline literature [37], computer aided geometric design (CAGD) [19], and computer graphics (e.g. PS/TT fonts) [26] but have hitherto not been widely adopted as a basis for high order finite element approximations.

One immediate benefit of using the Bernstein basis is the ease with which one can visualize and post-process finite element solutions owing to the ubiquitous usage of the Bernstein basis in CAGD and the computer graphics community. Generally, visualization and post-processing, including computing iso-surfaces and gradients, of a high order approximation is considerably more complicated than for a low order approximation [38]. For example, visualizing a high order approximation expressed using hierarchical bases typically requires the explicit evaluation of Jacobi polynomials at large number of points which can become prohibitively expensive [28]. Nevertheless, if the approximation is expressed in Bernstein-Bézier form, then techniques developed in the CAGD community enable one to visualize a degree $p$ approximation in $\mathcal{O}(p^3)$ operations as described in section 3.

The suitability of the Bernstein basis for finite element approximations is less clear-cut. Here, among other things, one needs to compute moments of the data with the basis functions (e.g. when constructing the load vector) which, along with the assembly of the element matrices can dominate the computational costs. In the case of tensor product elements, one can use the sum factorization approach, pioneered by Orszag [36], to efficiently compute matrix-vector products and, although less well-known, to evaluate moments of the data. Standard hierarchical bases on a triangle do not naturally have a tensorial structure and are therefore not amenable to sum factorization approaches. Tensorial hierarchical bases [17, 28] circumvent this difficulty, but lack rotational symmetry and are sub-optimal when it comes to the evaluation of the element matrices. Perhaps surprisingly, the Bernstein basis was shown in [2] to *naturally* have the tensorial property, which is needed for the sum factorization approach, despite having been known for decades prior to the realization of the importance of the tensorial property. We briefly discuss how the AAD algorithms [2] can be used to construct moments efficiently, and enable the evaluation of the element matrices in optimal complexity in section 3.5. In particular, we show that these algorithms can be used to calculate quantities of interest of the solution in $\mathcal{O}(p^3)$ operations per element.

The aforementioned computational properties of the Bernstein basis come at a price: the ill-conditioning of the resulting matrices. For example, the mass matrix for the Bernstein basis has a condition number which grows as $\mathcal{O}(2^{2p}p^{-1/2})$ [31], whereas the mass (and stiffness) matrix for hierarchical bases has condition numbers which grow at $\mathcal{O}(p^4)$ or faster as we increase the order [4, 32, 35]. Recently an Additive Schwarz Method (ASM) preconditioner for the mass matrix was developed [5] which results in a uniform $\mathcal{O}(1)$ bound on the growth of the condition number independent of the polynomial order $p$. The preconditioner is basis independent and therefore applies to both hierarchical and Bernstein bases. In section 4, we present algorithms which implement the ASM preconditioner in $\mathcal{O}(p^3)$ operations in the case of the Bernstein basis. We exploit a number of properties of the Bernstein basis to reduce computational costs. A key component of the algorithm is the static condensation of the interior degrees of freedom on each element: we present an algorithm which allows one to achieve this in $\mathcal{O}(p^3)$ operations.

In section 2, we present some canonical applications and use them to highlight some of the specific difficulties that one encounters when attempting to use a high order scheme to approximate their solutions. The above developments mean one can tackle each of these problems by using the Bernstein basis at an overall complexity of $\mathcal{O}(p^3)$ operations. Finally, in section 5 we return to the canonical examples described in section 2, and illustrate the performance of the above procedures when applied to these cases.

## 2. Model Problems, Finite Element Formulations, and Computational Challenges

We consider three prototypical problems which theory suggests should be amendable to high order FEM approximations yet each problem exhibits features which present challenges in terms of the efficient implementation of a high order scheme.

In each case $\Omega \subset \mathbb{R}^2$ is a polygonal domain which is partitioned into the

union $\mathcal{T}$ of non-overlapping triangular elements with the standard assumptions that the nonempty intersection of any two distinct elements from $\mathcal{T}$ is either a common vertex or a single common edge. More generally, we consider a family of partitions which is assumed to be shape regular in that there exists a number $c > 0$ such that for all partitions, each triangle $T$ contains an incircle with radius $r \geq h_T/c$ where $h_T$ is the diameter of $T$.

Let $\mathbb{P}_p(T) = \text{span}\{x^\alpha y^\beta : 0 \leq \alpha, \beta, \alpha + \beta \leq p\}$ denote the space of polynomials of total degree $p$ on $T \in \mathcal{T}$. Define the standard $H^1$-conforming finite element space $X = \{u \in H^1(\Omega) : u|_T \in \mathbb{P}_p(T), \forall T \in \mathcal{T}\}$ and the $H_0^1$-conforming space $X_0 = X \cap H_0^1(\Omega)$. Let $\{\varphi_i\}_{i=1}^N$ be a basis for $X$, so that any $u \in X$ or $X_0$ can be written as $u = \vec{u}^T\vec{\varphi}$ for $\vec{u} \in \mathbb{R}^N$, where $\vec{\varphi}$ is the vector whose components are the basis functions. Let $\mathbf{M}$ and $\mathbf{S}$ be the associated mass and stiffness matrices.

### *2.1. Sine-Gordon Equation*

The sine-Gordon equation arises in a range of applications, including differential geometry [48] and modeling the dislocation of crystals [9, 16], and consists of seeking $u$ such that

$$\frac{\partial^2 u}{\partial t^2} = \Delta u - \sin u, \quad (x, y) \in (-7, 7)^2, t > 0 \tag{1}$$

subject to initial conditions given, for example, by [11]

$$\begin{aligned} u(x, y, 0) &= u_0(x, y) = 4 \arctan \exp(x + 1 - 2\,\text{sech}(y + 7) - 2\,\text{sech}(y - 7)) \\ \tfrac{\partial}{\partial t} u(x, y, 0) &= w_0(x, y) = 0 \end{aligned}$$

along with homogeneous Neumann boundary conditions. The variational form of the problem consists of seeking $u(t) \in H^1(\Omega), t > 0$ such that

$$\frac{\partial^2}{\partial t^2}(u, v) = -(\nabla u, \nabla v) - (\sin u, v) \qquad \forall v \in H^1(\Omega) \tag{2}$$

where $u(0) = u_0$ and $\frac{\partial}{\partial t} u(0) = w_0$. The solution is smooth (see fig. 1), and thus should be amenable to approximation using higher order methods [41].

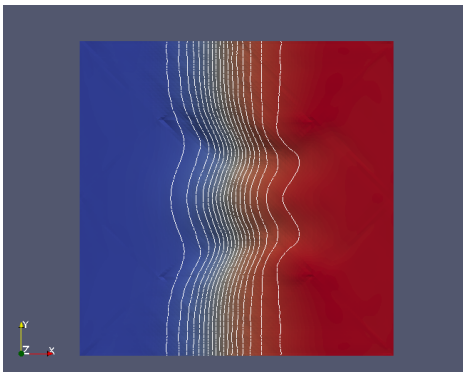


Figure 1: Contour plot of the solution of the sine-Gordon equation with the initial conditions from section 2.1 at $t = 5$.

Let $u_p(t) \in X$ be the Galerkin approximation to eq. (2) subject to the initial conditions $u_p(0) = u_{0p}$ and $\frac{\partial}{\partial t} u_p(0) = w_{0p}$ where $u_{0p}, w_{0p} \in X$ satisfies

$$\begin{aligned} (u_{0p}, v) &= (u_0, v) \qquad \forall v \in X \\ (w_{0p}, v) &= (w_0, v) \qquad \forall v \in X. \end{aligned} \tag{3}$$

Writing $u_p(t) = \vec{u}(t)^T \vec{\varphi}$ for $\vec{u}(t) \in \mathbb{R}^N$, the semi-discrete problem takes the form

$$\mathbf{M} \tfrac{\mathrm{d}^2}{\mathrm{d}t^2} \vec{u}(t) = -\mathbf{S}\vec{u}(t) - (\sin u_p(t), \vec{\varphi}).$$

A fully discrete scheme can be obtained by using a Nyström method [25, p. 285] to discretize the temporal derivative:

$$\begin{aligned} z &= \vec{u}^n \\ \mathbf{M}\vec{u}_1^{n+1} &= -\mathbf{S}\vec{w} - (\sin z, \vec{\varphi}) \\ z &= \vec{u}^n + (\Delta t)\vec{u}_t^n/2 + (\Delta t)^2 \vec{u}_1^{n+1}/8 \\ \mathbf{M}\vec{u}_2^{n+1} &= -\mathbf{S}\vec{w} - (\sin z, \vec{\varphi}) \\ z &= \vec{u}^n + (\Delta t)\vec{u}_t^n + (\Delta t)^2 \vec{u}_2^{n+1}/2 \\ \mathbf{M}\vec{u}_3^{n+1} &= -\mathbf{S}\vec{w} - (\sin z, \vec{\varphi}) \\ \vec{u}^{n+1} &= \vec{u}^n + (\Delta t)\vec{u}_t^n + (\Delta t)^2(\vec{u}_1^{n+1}/6 + \vec{u}_1^{n+2}/3) \\ \vec{u}_t^{n+1} &= \vec{u}_t^n + \Delta t(\vec{u}_1^{n+1}/6 + 2\vec{u}_2^{n+1}/3 + \vec{u}_3^{n+1}/6) \end{aligned} \tag{4}$$

where $\vec{u}^n = \vec{u}(n\Delta t)$, $\vec{\varphi}^T\vec{u}^0 = u_{0p}$, and $\vec{\varphi}^T\vec{u}_t^0 = w_{0p}$. Implicit time-stepping schemes will be considered later.

The first difficulty encountered in the implementation of eq. (4) is the computation of the nonlinear moment $(\sin z, \vec{\varphi})$ whose efficient computation is essential as it has to be evaluated at *every* sub-step. A straightforward treatment of the vector $(\sin z, \vec{\varphi})$ would entail using a quadrature rule with $\mathcal{O}(p^2)$ quadrature points for each of the $\mathcal{O}(p^2)$ entries incurring a cost of $\mathcal{O}(p^4)$ in basis function evaluations [2]. For most hierarchical bases, function evaluation involves evaluations of univariate Jacobi polynomials using a recursion at a cost of $\mathcal{O}(p)$ operations per point [1, 43]. It is possible to use precomputed arrays, in which the values of the basis functions at quadrature points are cached, but the current computing platforms lean towards the view that memory access is costlier than CPU cycles [39].

The second difficulty is that one needs to solve *three* systems involving the mass matrix $\mathbf{M}$ at each time-step. The mass matrices obtained using hierarchical bases have condition numbers which grow as $\mathcal{O}(p^4)$, or faster, even for tensor product elements [4, 32, 35]. In [4, 32], it was shown that applying diagonal scaling as a preconditioner for the mass matrix results in a reduction of the condition number to $\mathcal{O}(p^2)$. Nevertheless, there is a significant cost involved in solving systems involving the mass matrix. Fortunately an Additive Scharz Method (ASM) preconditioner was recently developed for the mass matrix which results in a uniformly bounded condition number independent of $p$ [5]. We shall pursue this further in section 4 when we consider how to address the efficient inversion of the mass matrix.

Quite apart from issues of conditioning, efficient iterative methods also require fast matrix-vector multiplication. There are two ways to compute matrix-vector products. The first is the explicit construction of the mass and stiffness matrices, which will incur a cost of $\mathcal{O}(p^6)$ basis function evaluations if performed in a naive fashion [2], and to then compute the matrix-vector products directly. The second way is to use a matrix-free approach which enables the computation of the matrix-vector product in $\mathcal{O}(p^3)$ provided that a tensorial basis is used on

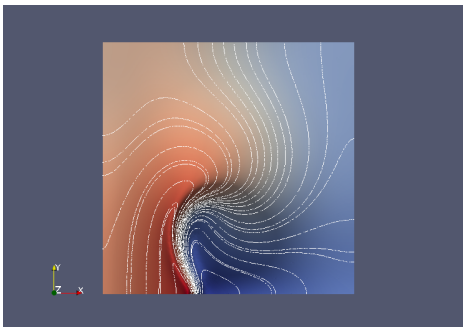


Figure 2: Contour plot of the solution of the $v$ component of the Brusselator equation with the initial conditions from section 2.2 at $t = 10$.

the triangle [28].

### *2.2. Brusselator*

The Brusselator system is a model of an autocatalytic chemical reaction [25, p. 248] and consists of seeking $(u(t), v(t)), t > 0$ such that

$$\begin{aligned} \frac{\partial u}{\partial t} &= 1 + u^2 v - 4.4u + 0.002\Delta u \\ \frac{\partial v}{\partial t} &= 3.4u - u^2 v + 0.002\Delta v \end{aligned} \qquad (x, y) \in (-1, 1)^2, \tag{5}$$

subject to homogeneous Neumann boundary conditions, and initial conditions given, for example, by

$$\begin{aligned} u(x, y, 0) &= u_0(x, y) = 0.5 + y \\ v(x, y, 0) &= v_0(x, y) = 1 + 5x. \end{aligned}$$

The corresponding variational formulation is to seek $u(t), v(t) \in H^1(\Omega), t > 0$ such that

$$\begin{aligned} \frac{\partial}{\partial t}(u, w) &= (1, w) + (u^2 v, w) - 4.4(u, w) - 0.002(\nabla u, \nabla w) \\ \frac{\partial}{\partial t}(v, w) &= 3.4(u, w) - (u^2 v, w) - 0.002(\nabla v, \nabla w) \end{aligned} \tag{6}$$

for all $w \in H^1(\Omega)$. Although the solution is smooth (see fig. 2), it does exhibits steep interior layers whose location changes as the solution evolves.

Let $u_p(t), v_p(t) \in X$ be the Galerkin approximations to eq. (6) for $u, v$ respectively, subject to initial conditions satisfying

$$\begin{aligned}(u_p(0), w) &= (u_0, w) \qquad \forall w \in X \\ (v_p(0), w) &= (v_0, w) \qquad \forall w \in X.\end{aligned}$$

Let $\vec{u}(t) \in \mathbb{R}^N$ be the vector such that $u_p(t) = \vec{u}(t)^T \vec{\varphi}$ and likewise for $\vec{v}(t) \in \mathbb{R}^N$, then the semi-discrete problem is

$$\begin{aligned}\mathbf{M}\frac{\partial}{\partial t}\vec{u}(t) &= (1, \vec{\varphi}) + (u_p^2(t)v_p(t), \vec{\varphi}) - 4.4\mathbf{M}\vec{u}(t) - 0.002\mathbf{S}\vec{u}(t) \\ \mathbf{M}\frac{\partial}{\partial t}\vec{v}(t) &= 3.4\mathbf{M}\vec{u}(t) - (u_p^2(t)v_p(t), \vec{\varphi}) - 0.002\mathbf{S}\vec{v}(t)\end{aligned}.$$

To arrive at the fully discrete scheme, we use an IMEX scheme [40] for the time discretization as follows:

$$\begin{aligned}\frac{\mathbf{M}\vec{u}^{n+1} - \mathbf{M}\vec{u}^n}{\Delta t} &= (1, \vec{\varphi}) + (u_n^2 v_n, \vec{\varphi}) - 4.4\mathbf{M}\vec{u}^{n+1} - \frac{0.002}{2}(\mathbf{S}\vec{u}^{n+1} + \mathbf{S}\vec{u}^n) \\ \frac{\mathbf{M}\vec{v}^{n+1} - \mathbf{M}\vec{v}^n}{\Delta t} &= 3.4\mathbf{M}\vec{u}^n - (u_n^2 v_n, \vec{\varphi}) - \frac{0.002}{2}(\mathbf{S}\vec{v}^{n+1} + \mathbf{S}\vec{v}^n)\end{aligned} \tag{7}$$

where $\vec{u}^n$ is the approximation at $n\Delta t$, and $(u_n^2 v_n, \vec{\varphi})$ is shorthand for the nonlinear term $(u_p^2(t)v_p(t), \vec{\varphi})$ at time $t = n\Delta t$. Observe that if we were to use a fully explicit scheme, the CFL condition for stability is $\Delta t \leq C\frac{h^2}{p^4}$ which, owing to the rapid decrease with $p$, is generally regarded as being overly restrictive for practical computations. Instead, one typically sees $\Delta t \sim \frac{h^2}{p^2}$ being used in practice in conjunction with an implicit scheme.

The efficient application of a high order scheme to the solution of the Brusselator system encounters all of the difficulties which we noted for the sine-Gordon equation. In addition, the Brusselator system involves the repeated inversion of the matrices $\mathbf{M} + 0.001\Delta t\mathbf{S}$ and $5.4\mathbf{M} + 0.001\Delta t\mathbf{S}$, as opposed to the pure mass matrix. Previously, we alluded the availability of an ASM preconditioner for the mass matrix. Can this preconditioner for the pure mass matrix play a useful role in the case of implicit schemes?

The two dimensional version of Schmidt's inequality [15] implies there is a

constant $c$, independent of $h$ and $p$, such that $0 \leq \mathbf{S} \leq c\frac{p^4}{h^2}\mathbf{M}$, hence we have

$$\mathbf{M} \leq \mathbf{M} + 0.001\Delta t\mathbf{S} \leq \left(1 + c\frac{p^4\Delta t}{h^2}\right)\mathbf{M}.$$

Let $\mathbf{P}^{-1}$ denote the uniform preconditioner for the mass matrix described in [5]. Then, using $\mathbf{P}^{-1}$ to precondition the implicit scheme gives a condition number satisfying

$$\kappa(\mathbf{P}^{-1}(\mathbf{M} + 0.001\Delta t\mathbf{S})) \leq C\frac{p^4\Delta t}{h^2}.$$

Observe that if one uses a time step which satisfies the CFL condition for the explicit scheme (i.e. $\Delta t \sim \frac{h^2}{p^4}$), then the condition number will be uniformly bounded. Alternatively, taking a step size of $\Delta t \sim \frac{h^2}{p^2}$ results in the condition number of the operator growing as $\mathcal{O}(p^2)$. Therefore a preconditioner for the mass matrix also provides a useful preconditioner for the systems arising from an implicit time stepping scheme.

Finally, a difficulty (pertinent also to the case of the sine-Gordon equation) which often remains unacknowledged in high order finite elements analysis is the cost of post-processing and visualization of the resulting finite element solution. A straightforward approach to visualization based on evaluating the solution at sufficiently many points and using a standard graphics package, would require the evaluation of the solution at $\mathcal{O}(p^2)$ points. At each of those points, we need to evaluate the solution (a vector with with $\mathcal{O}(p^2)$ entries) meaning a total of $\mathcal{O}(p^4)$ Jacobi polynomial evaluations are needed to evaluate $u$. The same costs apply if one wishes to visualize a component of the gradient etc. We discuss the issue of visualization and post-processing in section 3.

### *2.3. Problems Exhibiting Boundary Layers*

Let $0 < \varepsilon \ll 1$ be a parameter, and consider the problem on $\Omega = (0,1)^2$

$$\begin{aligned} u - \varepsilon^2\Delta u &= f \qquad x \in \Omega \\ u &= 0 \qquad x \in \partial\Omega \end{aligned}$$

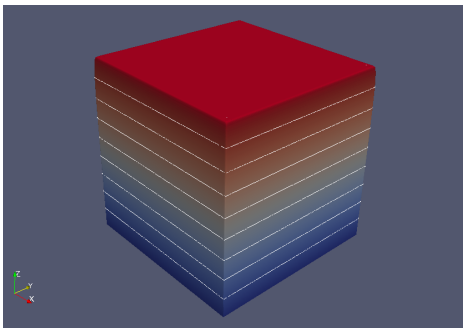


Figure 3: Contour plot of the solution of the singularly perturbed problem with $\varepsilon^2 = 10^{-3}$ and $f = 1$.

where $f \in L^2(\Omega)$. This is an example of a singularly perturbed problem in which the solution exhibits steep layers of width $\mathcal{O}(\varepsilon)$ in the neighborhood of the boundary [34]; see fig. 3 for a plot of the solution for $\varepsilon^2 = 10^{-3}$ with $f = 1$. This problem serves as a prototype for a large class of problems arising in mechanics including, for example, the linear elastic response of thin bodies [24].

The variational form consists of seeking $u \in H_0^1(\Omega)$ such that

$$(u, v) + \varepsilon^2(\nabla u, \nabla v) = (f, v) \qquad \forall v \in H^1(\Omega).$$

In fully discrete form, we arrive at the linear system

$$(\mathbf{M} + \varepsilon^2 \mathbf{S})\vec{u} = \vec{f} \tag{8}$$

where $\vec{f} = (f, \vec{\varphi})$. Whilst the operator $\mathbf{M} + \varepsilon^2\mathbf{S}$ has, at first glance, the same structure as the operators which arose in the Brusselator example, viz $\mathbf{M} + c\Delta t\mathbf{S}$, the present case poses an additional layer of difficulty which we shall now explain.

The anisotropic behavior of the solution in the neighborhood of the boundary means that, in order to obtain a robust scheme in $\varepsilon$, anisotropic or stretched elements should be used at the boundary in conjunction with regular elements on the interior [6]. Moreover, whilst the solution has boundary layers, it is analytic and as such high order methods can exhibit exponential rates of convergence

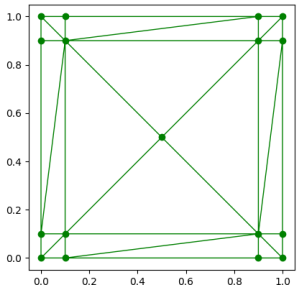



Figure 4: Plot of the mesh to approximate the boundary layer problem section 2.3. The needle elements around the boundaries have thickness of $p\varepsilon$ in order to resolve the rapid changes [34, 42].

provided that the anisotropy of the elements is properly combined with the polynomial order $p$ [42].

The correct combination of anisotropic and $p$ consists of using anisotropic elements of width $\mathcal{O}(p\varepsilon)$ along the boundary as illustrated in fig. 4 [42]. This approach gives *robust* exponential convergence with respect to $\varepsilon$ and, as such, will outperform a pure $h$-version or pure $p$-version method [34]. Of course, using a single layer of anisotropic elements around the boundary means that we drop our earlier assumption that the family of partitions is shape uniform.

The fresh computational issue that arises is that the aspect ratio of anisotropic elements has a detrimental effect on the conditioning of the stiffness matrix $\mathbf{S}$ [29] resulting in issues with iterative solvers [34]. Existing preconditioners for anisotropic elements are either inapplicable to the meshes from [34, 42] described above [46] or give condition numbers dependent on the factor $\varepsilon$ [33].

The above difficulties notwithstanding, we again propose to simply use the mass matrix preconditioner $\mathbf{P}^{-1}$ from [5] to precondition the systems arising from meshes such as the one shown in fig. 4. A scaling argument applied to the usual two dimensional Schmidt's inequality [15] on isotropic elements can

be used to deduce that

$$\mathbf{M} \leq \mathbf{M} + \varepsilon^2 \mathbf{S} \leq \left(1 + c\varepsilon^2 \frac{p^4}{p^2 \varepsilon^2}\right) \mathbf{M} \tag{9}$$

$$\leq (1 + cp^2)\mathbf{M}. \tag{10}$$

Consequently, using the mass preconditioner $\mathbf{P}^{-1}$ from [5], we have $\kappa(\mathbf{P}^{-1}(\mathbf{M} + \varepsilon^2 \mathbf{S})) \leq Cp^2$ with $C$ independent of $p$, $\varepsilon$ and the number of elements. The key advantage of using the mass matrix is that the condition number is *independent of* $\varepsilon$ whereas alternative approaches result in a condition number depending on $\varepsilon^{-1} \gg 1$.

### *2.4. Summary*

In summary, applying high order methods to tackle the above prototypical problems encounters the following challenges:

1. Calculation of the nonlinear moments, such as $(u_n^2 v_n, \vec{\varphi})$ and $(\sin z, \vec{\varphi})$, is potentially inefficient. As discussed previously, bases which can utilize the sum factorization technique are adept at computing moments and matrix-vector products. In section 3.5, we will briefly discuss how the Bernstein polynomials can use algorithms presented in [2] to calculate the nonlinear moments and the residuals in $\mathcal{O}(p^3)$ operations.

2. Transient problems will require the use of a time stepping scheme, which results in the need to invert either the mass matrix or a perturbation thereof. We propose to solve such systems efficiently by implementing the ASM preconditioner developed in [5] using the Bernstein basis in section 4. Furthermore, the foregoing discussion showed for the treatment of the matrices arising in implicit time stepping schemes and from problems where anisotropic elements are used, preconditioning the mass matrix can be an effective approach.

3. Finally, once the simulation is complete, one typically wishes to either visualize the solution or carry out post-processing to calculate quantities

of interest. We will exposit algorithms which can easily visualize and post-process the solutions obtained using the Bernstein basis in section 3.

## 3. Visualization and Post-Processing

Bernstein-Bézier polynomials have played a fundamental role in the development of computer graphics, splines, PS/TT fonts and computer-aided geometric design (CAGD), resulting in a wealth of elegant and effective algorithms for the visualization and graphical post-processing of polynomials written in Bernstein form [18, 20]. In this section, we formally introduce the Bernstein polynomials and give a brief overview of efficient $\mathcal{O}(p^3)$ algorithms for the implementation of post-processing procedures which are pertinent to finite element analysis (e.g. point evaluation, visualization, and evaluations of quantities of interest).

### *3.1. Bernstein Polynomials*

Let $T$ be a non-degenerate triangle in $\mathbb{R}^2$ with vertices $v_1, v_2, v_3$. For a fixed integer $p \geq 3$, we define the domain points as

$$\mathcal{D}^p(T) = \left\{ \frac{1}{p} \left( \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 \right) : (\alpha_1, \alpha_2, \alpha_3) \in \mathcal{I}^p \right\}$$

where the index set $\mathcal{I}^p = \{\alpha := (\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}_+^3 : \sum_{k=1}^3 \alpha_k = p\}$. It is natural to classify the domain points into vertices, edges or interior points. The interior domain points are those associated with $\alpha \in \mathcal{I}^p$ with strictly positive components. The vertex domain points are associated with the indices $(p, 0, 0), (0, p, 0)$ and $(0, 0, p)$. Finally, the edge domain points are the remaining domain points; see fig. 5.

The barycentric coordinates $\lambda_i \in \mathbb{P}_1(T), i \in \{1, 2, 3\}$ of $T$ are affine functions such that $\lambda_i(v_j) = \delta_{ij}$ for $i, j \in \{1, 2, 3\}$. The bivariate Bernstein polynomials of degree $p$ associated with triangle $T$ are then defined by

$$B_\alpha^p = \frac{p!}{\alpha_{1!}\alpha_{2!}\alpha_{3!}} \lambda_1^{\alpha_1} \lambda_2^{\alpha_2} \lambda_3^{\alpha_3}, \qquad \alpha \in \mathcal{I}^p.$$

There is a natural one-to-one correspondence between Bernstein polynomials, domain points and the index set $\mathcal{I}^p$. Every Bernstein polynomial on a triangle

can be readily classified as an interior, an edge or a vertex polynomial in much the same way as domain points. We denote by $B_V^p, B_E^p, B_I^p$ as the sets of all vertex, edge and interior Bernstein polynomials; see fig. 5.

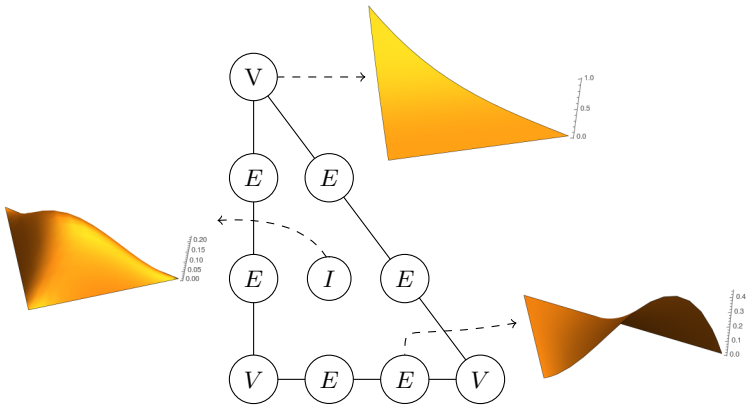



Figure 5: Figure showing domain points for degree $p = 3$ along with some plots of a typical Bernstein polynomial corresponding to the domain points. V stands for vertex, E stands for edge and I stands for interior.

Every polynomial $u \in \mathbb{P}_p(T)$ can be expressed in terms of degree $p$ Bernstein polynomials:

$$u = \sum_{\alpha \in \mathcal{I}^p} c_\alpha^p B_\alpha^p$$

the so-called B-form of $u$. The coefficients $c_\alpha^p$ are usually referred to as the B-net or control points by the graphics community [18].

Likewise, one can define the univariate Bernstein polynomials: let $\lambda_1, \lambda_2$ be the barycentric coordinates of a point on the interval $[a, b]$, then the univariate Bernstein polynomials of degree $p$ on the interval are defined as

$$B_i^p = \binom{p}{i} \lambda_1^i \lambda_2^{p-i}, \qquad i = 0, \ldots, p.$$

*3.2. Point Evaluation using de Casteljau Algorithm*

The de Casteljau algorithm is an elegant and stable recursive scheme for the evaluation of a polynomial written in terms of Bernstein polynomials [18]. Given

the control points $\{c_\alpha^p\}$ of a polynomial $u$ in B-form, we fix a point $P \in T$ at which we want to evaluate $u$, and let $\lambda_1, \lambda_2, \lambda_3$ be the values of the barycentric coordinates of $P$. The de Casteljau algorithm consists of recursively defining points $\{c_\beta^k\}$ for $k \in [0, \dots, p-1]$ and $\beta \in \mathcal{I}^k$ by

$$c^k_{(\beta_1,\beta_2,\beta_3)} := \lambda_1 c^{k+1}_{(\beta_1+1,\beta_2,\beta_3)} + \lambda_2 c^{k+1}_{(\beta_1,\beta_2+1,\beta_3)} + \lambda_3 c^{k+1}_{(\beta_1,\beta_2,\beta_3+1)}.$$

The recursion terminates with a single coefficient $c^0_{000}$ at a cost of $\mathcal{O}(p^3)$ operations which, remarkably, coincides $u(P)$; see fig. 6 for an example in the case $p = 3$.

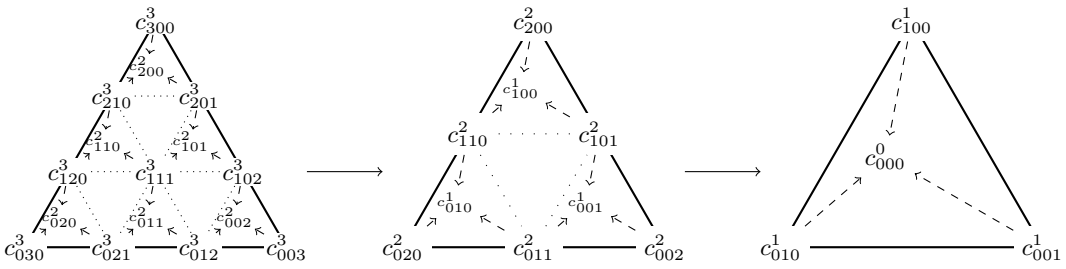



Figure 6: Example of applying de Casteljau algorithm to $p = 3$ case

The de Castlejau algorithm is the archetypal example of a *pyramid* algorithm [22]. Specifically, if we stack the coefficients appearing in fig. 6 as shown in fig. 7, in which each layer corresponds to the recursion level, we obtain a pyramid of coefficients with $c_\alpha^p$ on the bottom and $c^0_{000}$ at the summit.

A key property of the de Castlejau algorithm is that the coefficients which emerge on its three vertical faces of the pyramid satisfies the *blossoming* property. In order to explain what this means, we first label the vertices corresponding to domain points $c^3_{300}, c^3_{030}, c^3_{003}$ (i.e. the vertices of the triangle) as $A, B, C$ respectively and again fix the point $P \in T$ corresponding to barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$ as before (recall $u(P) = c^0_{000}$).

Blossoming is the property whereby the B-form polynomial defined by the coefficients laid out in Triangle 1 in fig. 8 (the left face in fig. 7) equals the

restriction of $u$ to the region $\triangle ABP$, i.e.

$$u_1|_{\triangle ABP} = u|_{\triangle ABP}.$$

The same property holds true for $\triangle BCP$ with coefficients as in Triangle 2, and for $\triangle ACP$ with coefficients from Triangle 3. In other words, we have

$$u(x)|_{\triangle ABC} = \begin{cases} u_1(x) & x \in \triangle ABP \\ u_2(x) & x \in \triangle BCP \\ u_3(x) & x \in \triangle ACP \end{cases}.$$

A pseudo-code implementation of de Casteljau algorithm with the blossoming coefficients stored can be seen in algorithm 1. Note that the blossoms for the faces are a natural by-product of applying the de Casteljau algorithm.

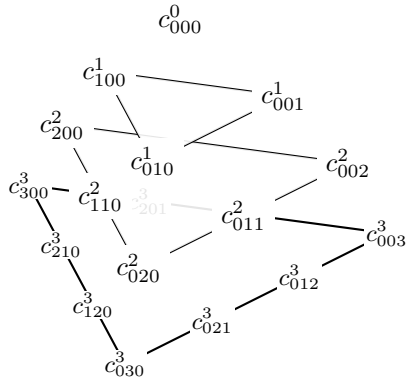
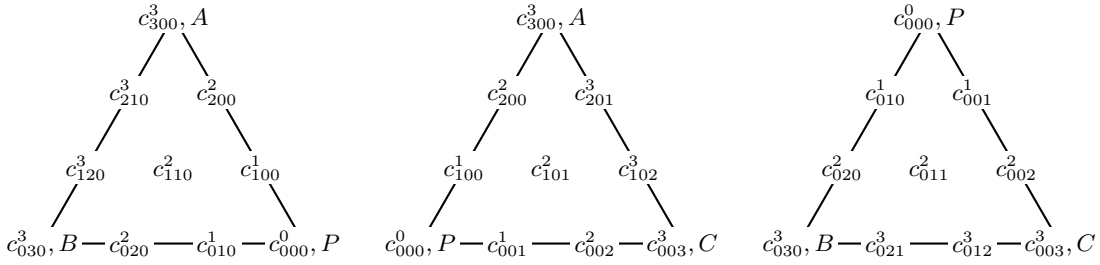



Figure 7: Rearranging the de Casteljau algorithm to a pyramid in the $p = 3$ case. The interior coefficients (such as $c^3_{111}$) are left out for clarity.

### *3.3. Visualization*

While the de Casteljau algorithm is stable, using it to evaluate large numbers of points for plotting is not an efficient strategy (e.g $\mathcal{O}(p^3)$ operations are required for reach of the $\mathcal{O}(p^2)$ points netting an overall cost of $\mathcal{O}(p^5)$). Consequently, a standard technique to the rendering of Bernstein-Bézier surfaces in the computer graphics community consists of plotting the surface obtained




Triangle 1: $u_1(x)$ Triangle 2: $u_2(x)$ Triangle 3: $u_3(x)$

Figure 8: Example of blossoming

by linearly interpolating the coefficients $\{c^p_\alpha\}$ of the Bernstein polynomial [18]. This B-net is a convex hull for the polynomial, and approximates the surface. If higher resolution is needed than provided by the original B-net of the solution, then the *subdivision* algorithm can be invoked to create a finer net which then can be rendered in the same fashion.

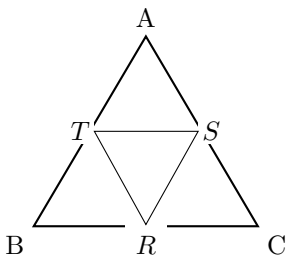



Figure 9: The subdivision algorithm: given the control points on $\triangle ABC$, we divide it into four triangles $\triangle ATS$, $\triangle TRS$, $\triangle TBR$, and $\triangle SRC$ whose control points equals the same polynomial.

The subdivision algorithm consists of dividing the original triangle into four triangles representing the same polynomial[2]; see fig. 9. Although the original function remains unchanged, the B-net representing $u$ now contains roughly

[2]We note that the de Casteljau algorithm with blossoming coefficients divides the triangle into three triangles representing the same polynomial (assuming the point lies in the interior of the triangle).

**Algorithm 1** de Casteljau Algorithm (with blossoming coefficients)

**Require:** **abc** 2D array of the B-net of the polynomial of degree $p$

1: **function** DECAST($\lambda_1, \lambda_2, \lambda_3, \mathbf{abc}$)
2: $\mathbf{apc}, \mathbf{abp} := \text{zeros}((p+1, p+1))$ ▷ Stores blossoming data
3: $\mathbf{apc} = \mathbf{abc}[:, 0]$ ▷ Store first rows before overwriting
4: $\mathbf{abp} = \mathbf{abc}[0, :]$
5: **for** $k = 0, \ldots, p-1$ **do**
6: **for** $i = 0, \ldots, p-k-1$ **do**
7: **for** $j = 0, \ldots, i$ **do** ▷ de Casteljau step
8: $\mathbf{abc}[j, i-j] = \lambda_1 \mathbf{abc}[j, i-j] + \lambda_2 \mathbf{abc}[j+1, i-j] + \lambda_3 \mathbf{abc}[j, i-j+1]$
9: **end for**
10: **end for**
11: $\mathbf{apc}[0 : p-k, k+1] = \mathbf{abc}[0 : p-k, 0]$ ▷ Store the blossoming coefficients before progressing to the next level
12: $\mathbf{abp}[k+1, 0 : p-k] = \mathbf{abc}[0, 0 : p-k]$
13: **end for**
14: ▷ **apc** and **abp** contains the B-net for triangles $APC$ and $ABP$ respectively. See fig. 8.
15: **return** $\mathbf{abc}, \mathbf{apc}, \mathbf{abp}$ ▷ **abc** contains the coefficients for **pbc**
16: **end function**

four times as many B-net points obtained at the cost of just four applications of the de Casteljau algorithm with storage of the blossoming coefficients (see algorithm 2) [21, §8.1]; see table 1 for a comparison in the cost of visualization in terms of the number of operations needed per point when using de Casteljau algorithm versus the subdivision algorithm.

The subdivision algorithm converges quadratically in the number of subdivision levels $\ell$, in the sense that for a given triangle $T$ with diameter $h_T$, the

Table 1: Table to illustrate the benefit of using subdivision algorithm by displaying the cost per point of visualization assuming $\mathcal{O}(1)$ number of subdivisions; typically, only two or three subdivisions are needed for visual fidelity.

| **Method** | **Number of Points** | **Cost per Point** | **Total Cost** |
|---|---|---|---|
| de Casteljau | $\mathcal{O}(p^2)$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^5)$ |
| Subdivision | $\mathcal{O}(p^2)$ | $\mathcal{O}(p)$ | $\mathcal{O}(p^3)$ |

error at the $\ell$th level of subdivision is

$$\|u - \bar{u}_\ell\|_\infty \leq \frac{C}{2^\ell} \|\Delta u\|_\infty$$

for $C$ independent of $h$, where $u \in \mathbb{P}_p(T)$ and $\bar{u}_\ell$ is the linear interpolation of the $\ell$th level subdivided B-net [12, 13]. In fig. 10, we plot a B-net and its subdivisions; we observe that two or three subdivisions is usually more than enough for visual fidelity.

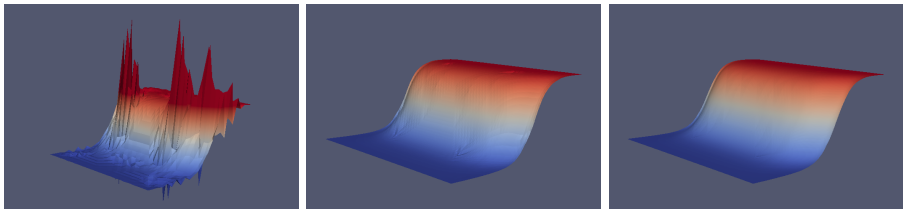
Once the subdivision step is completed, one can save the resulting B-net and the domain points as VTK files in order that sophisticated visualization software akin to Paraview or VisIt can easily process it. From here, robust algorithm in those software packages can post-process the approximation including plotting contour lines.

**Algorithm 2** Subdivision Algorithm on a single triangle

**Require:** **abc** array of the coefficients of B-form polynomial of degree $p$

1: **function** SUBDIVISION(**c**)
2: $\_, \mathbf{abr}, \mathbf{arc} = \texttt{decast}(0, .5, .5, \mathbf{abc})$ ▷ We use the blossoming coefficients from algorithm 1; the $\_$ notation means we do not use the result.
3: $\mathbf{tbr}, \_, \_ = \texttt{decast}(.5, .5, 0, \mathbf{abr})$ ▷ Obtain triangle $TBR$ from fig. 9
4: $\mathbf{src}, \mathbf{ars}, \_ = \texttt{decast}(.5, 0, .5, \mathbf{arc})$ ▷ Obtain triangle $SRC$ from fig. 9
5: $\mathbf{trs}, \_, \mathbf{ats} = \texttt{decast}(1, 1, -1, \mathbf{ars})$ ▷ point $T$ is outside of $\triangle ARS$; this step gives us the last two triangles of the subdivision.
6: **return ats, trs, tbr, src**
7: **end function**

Figure 10: Figures showing the refinements of the subdivision algorithm for $p = 12$ and 16 elements on the square for the initial condition of the sine-Gordon example. In general, the number of refinements needed is only 2 or 3 depending on the order and size of the elements.



(a) 0 subdivisions (b) 1 subdivision (c) 2 subdivisions

The efficient rendering of the B-net can be accomplished using OpenGL "evaluators" (see `glEvalMesh2` in [44]). These OpenGL evaluators are defined on a rectangular patch but a simple transformation of the coefficients on a triangle to the rectangle can be employed [27, 50]. Unfortunately, in many implementations, there is vendor and hardware dependent constant, namely `GL_MAX_EVAL_ORDER`, which sets the maximum order that OpenGL can plot, which is often set to just $p < 8$.

### *3.4. Computation and Visualization of Gradients and Higher Derivatives of the Solution*

We now describe how to easily compute and visualize the *gradient* and *higher order derivatives* from a B-form polynomial. The gradient of a Bernstein polynomial can be expressed as a sum of Bernstein polynomials

$$\nabla B^p_\alpha = p \sum_{k=1}^{3} B^p_{\alpha - e_k} \nabla \lambda_k \tag{11}$$

where the sum is over when $\alpha - e_k$ is a valid multi-index [3]. Let $u$ be a given polynomial expressed in B-form, then by eq. (11)

$$\nabla u = \sum_{\alpha \in \mathcal{I}^p} c^p_\alpha \left( p \sum_{k=1}^{3} B^p_{\alpha - e_k} \nabla \lambda_k \right) = \sum_{\beta \in \mathcal{I}^{p-1}} \vec{c}^{\,p-1}_\beta B^{p-1}_\beta$$

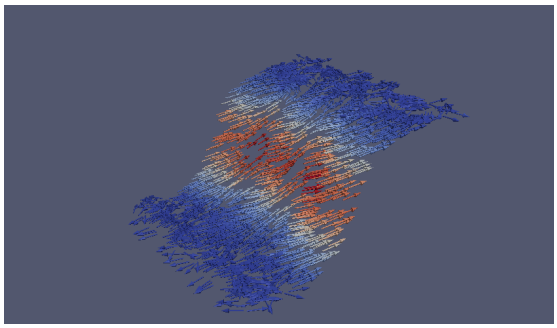


Figure 11: Example vector plot of gradients for the sine-Gordon example at $t = 5$ with 2 subdivisions. Note that the gradient glyphs are superimposed over the plot of the sine-Gordon solution.

where

$$\vec{c}_\beta^{\,p-1} = p \sum_{k=1}^{3} c_{\beta+e_k} \nabla \lambda_k \qquad \forall \beta \in \mathcal{I}^{p-1}.$$

Hence, to compute the gradient, we compute $\vec{c}_\beta^{\,p-1}$ from $c_\alpha^p$ at a cost of $\mathcal{O}(p^2)$ operations. One can then use the subdivision algorithm on $\vec{c}_\beta^{\,p-1}$ componentwise to plot the gradient. This results in a smaller B-net than by plotting the function values (i.e. $\alpha \in \mathcal{I}^p$ but $\beta \in \mathcal{I}^{p-1}$).

In order to obtain the B-net of the gradient on the same set of control points as the original approximate $u$, one would apply the Bernstein 2D degree raising algorithm (algorithm 14) component-wise which allows one to express $\vec{c}_\beta^{\,p-1}$ for $\beta \in \mathcal{I}^{p-1}$ as $\vec{c}_\alpha^{\,p}$ for $\alpha \in \mathcal{I}^p$; the cost of degree raising is $\mathcal{O}(p^2)$. See fig. 11 for an example of superimposing the gradients over the solution, and algorithm 3 for the general visualization algorithm. In fact this procedure can be generalized to arbitrary order derivatives; for example, to visualize the Hessian and superimpose it on the solution, one would have to first calculate the coefficients $\mathbf{c}_\beta^{p-2}$ appropriately, degree raise twice, then use the same subdivision algorithm component-wise.

**Algorithm 3** Function and Gradient Plotting Algorithm

**Require:** $c_\alpha^p$ coefficients of Bernstein polynomials

1: **function** VISUALIZE($c_\alpha^p$)
2: Calculate $\vec{c}_\beta^{\,p-1}$ from $c_\alpha^p$
3: $\vec{c}_\alpha^{\,p} = \texttt{DegreeRaise2D}(\vec{c}_\beta^{\,p-1})$ ▷ Perform degree raising component-wise
4: Apply `subdivision` to $c_\alpha^p$ and component-wise to $\vec{c}_\alpha^{\,p}$
5: Plot the resulting Bézier net from the subdivision algorithms
6: **end function**

*3.5. Evaluation of Quantities of Interest and Nonlinear Moments*

In many practical problems, the quantity of interest is not point values but rather some integral quantity of the solution $u$ such as the $L^2$ energy $\int_\Omega u^2\,dx$, $H^1$ energy $\int_\Omega (u^2 + |\nabla u|^2)\,dx$, the average displacement $\frac{1}{|\Omega|}\int_\Omega u\,dx$ etc [7]. In general, quantities of interest can be expressed as

$$Q[u] = \int_\Omega \eta(u, \nabla u)\,dx$$

for $\eta$ a given, possibly non-linear, function.

The quantity $Q[u]$ can be computed efficiently by exploiting the tensorial nature of the Bernstein basis (Lemma 1 of [2]). Given the control points $c_p^\alpha$ of the approximate $u$, we can apply algorithm 1 and eq (3.6) of [2] to directly compute $Q[u]$ for each element at a cost of $\mathcal{O}(p^3)$ operations. Furthermore, the tensorial property allows one to compute the residual and matrix-vector products in $\mathcal{O}(p^3)$ also. The following theorem from [2] is the key:

**Theorem 3.1.** *In two dimensions, the nonlinear moments*

$$\vec{\mu}_T(u, f) = \int_T B_\alpha^p(x) f(x, u, \nabla u)\,dx \qquad \forall \alpha \in \mathcal{I}^p$$

*where $T$ is a simplex and $f$ is an arbitrary nonlinear function can be computed with a cost of $\mathcal{O}(p^3)$.*

Here, we want to emphasize that theorem 3.1 allows us to calculate the nonlinear evaluation such as $(\sin z, \vec{\varphi})$ or $(u_n^2 v_n, \vec{\varphi})$ from section 2 in $\mathcal{O}(p^3)$. It is a straightforward application of the algorithm in Corollary 3 of [2].

Furthermore, we can calculate matrix-vector multiplication using $\mu_T$; for example, the mass matrix product can be calculated as

$$\vec{\mu}_T(u,1) = \int_T B^p_\alpha(x)u(x)\,dx = \int_T B^p_\alpha\big(\sum_{\beta\in\mathcal{I}^p} c^p_\beta B^p_\beta\big)\,dx = (\mathbf{M}\vec{u})_\alpha \qquad \forall\alpha\in\mathcal{I}^p$$

where $(\mathbf{M}\vec{u})_\alpha$ is the column corresponding to row $\alpha$. We refer to [2] for efficient techniques for the evaluation of matrix-vector products against the stiffness matrix etc.

## 4. Linear Solver and Preconditioning

In section 3, we discussed computation of the residual and visualization in $\mathcal{O}(p^3)$ using the Bernstein basis; all that remains is inverting the mass matrix (or a small perturbation thereof) in order to time-step. An unfortunate fact of the Bernstein basis is that its mass matrix condition number is $\mathcal{O}(2^{2p}p^{-1/2})$ [31]; an iterative solver will struggle, and direct solvers will lose many digits of accuracy. In this section, we present the implementation of a *uniform* preconditioner in both $h$ and $p$ [5] for the Bernstein basis mass matrix with a cost of $\mathcal{O}(p^3)$ operations.

We claim that we can simulate and post-process transient problems using an explicit time-stepper (e.g. section 2.1) in $\mathcal{O}(p^3)$ operations. Recall the error at iteration $n$ for conjugate gradient is bounded by

$$\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n \|\vec{e}_0\|$$

where $\kappa$ is the condition number of the preconditioned system and $\vec{e}_0$ is the initial residual vector [23, p. 636]. As the condition number $\kappa$ of the preconditioned mass matrix is bounded uniformly, the number of iterations needed by conjugate gradient to converge to a given tolerance $\varepsilon$ is also bounded uniformly by a constant $K$ independent of $p$ and $h$

$$K \le \log\frac{\varepsilon}{\|\vec{e}_0\|} \Big/ \log\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$$

Hence, to time-step up to time $T$ using an $\ell$ step explicit time-stepper with $\Delta t$ would require

$$\frac{T}{\Delta t}\ell N \cdot \mathcal{O}(p^3) \to \mathcal{O}(p^3)$$

operations *total*, including post-processing procedures.

### *4.1. Jacobi Polynomials*

We use the standard definition of 1D Jacobi polynomial [1] for $P_p^{(a,b)}$ where $p$ is the order and $a, b > -1$ are the weights. The orthogonality property is such that

$$\int_{-1}^{1} (1-x)^\alpha (1+x)^\beta P_m^{(\alpha,\beta)}(x) P_n^{(\alpha,\beta)}(x)\, dx \\ = \frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1} \frac{(n+\alpha)!(n+\beta)!}{(n+\alpha+\beta)!n!}\delta_{nm}.$$

A key identity relating Jacobi polynomials and 1D Bernstein polynomials together is

$$P_p^{(a,b)} = \sum_{i=0}^{n} \frac{\binom{p+a}{i}\binom{p+b}{p-i}}{(-1)^{p-i}\binom{p}{i}} B_i^p. \tag{12}$$

### *4.2. Additive Schwarz Preconditioner for Mass Matrix*

We present a short review of the additive Schwarz preconditioner presented in [5] for the mass matrix on a triangulation $\mathcal{T}$ of the domain $\Omega$. Let $X = \{u \in H^1(\Omega) : u|_K \in \mathbb{P}_p(K), \forall K \in \mathcal{T}\}$. Define $X_{K,I} := \mathbb{P}_p(K) \cap H_0^1(K)$ which is the space of polynomial bubble functions on element $K$, and let $X_I = \cup_{K\in\mathcal{T}} X_{K,I}$. We note that the interior Bernstein polynomials $B_I^p$ is a basis for $X_I$.

For each edge $e \in \mathcal{T}$, let $K_i$ be the elements such that $e \in \partial K_i$. Let

$$X_e := \operatorname{span}\{B_\alpha^p : \alpha \text{ domain points strictly in } e\}$$

and define the edge spaces

$$\widetilde{X}_e := \left\{u \in X_e : (u, w) = 0 \ \forall w \in X_I\right\}.$$

A key property is that each element in $\widetilde{X}_e$ and $X_e$ can be *uniquely determined by its value restricted to* $e$ (see Lemma 5.1 of [5]).

For $x \in [-1, 1]$, let

$$\nu(x) = \frac{(-1)^{\lfloor p/2 \rfloor + 1}}{\lfloor p/2 \rfloor} \frac{1-x}{2} P^{(1,1)}_{\lfloor p/2 \rfloor - 1}(x). \tag{13}$$

For each vertex $v \in \mathcal{T}$, let $K_i$ be the elements such that $v \in \partial K_i$, let $\lambda_i$ be the barycentric coordinate of $K_i$ such that $\lambda_i(v) = 1$. Define

$$\varphi_v(x) = \begin{cases} \nu(1 - 2\lambda_i) & x \in \cup_i K_i \\ 0 & \text{else} \end{cases}$$

which has the property that $\varphi_v(v) = 1$, and $\varphi_v(x) = 0$ for all $x \in \Omega \setminus \cup_i K_i$. Furthermore, $\varphi_v(x)$ on the edges on which it is supported is a scaling of $\nu$. Now define

$$X_V := \operatorname{span}\{\varphi_v : v \in \mathcal{T}\}$$

and

$$\widetilde{X}_V := \{u \in X_V : (u, w) = 0 \ \forall w \in X_I\}$$

Similar to $\widetilde{X}_e$, the space $\widetilde{X}_V$ is uniquely determined by the values on the vertices.

We can decompose $X$ as

$$X = X_I \oplus \widetilde{X}_V \oplus \bigoplus_{e \in \mathcal{T}} \widetilde{X}_e.$$

We now define the bilinear form on the subspaces in the decomposition:

- Interior space $X_I$:

$$a_I(u, w) := (u, w), \qquad u, w \in X_I.$$

- Vertex space $\widetilde{X}_V$:

$$a_V(\widetilde{u}, \widetilde{w}) := \frac{1}{p^4} \sum_{v \in \mathcal{T}} c_v \widetilde{u}(v) \widetilde{w}(v), \qquad \widetilde{u}, \widetilde{w} \in \widetilde{X}_V.$$

  where $c_v = \sum_{K_i} \frac{\text{area}(K_i)}{2}$ where $K_i$ are the elements such that $v \in \partial K_i$.

- Edge spaces $\widetilde{X}_e$ for all $e \in \mathcal{T}$:

$$a_e(\widetilde{u}, \widetilde{w}) := c_e \sum_{n=0}^{p-2} q_n \mu_n(\widetilde{u}) \mu_n(\widetilde{w}), \qquad \widetilde{u}, \widetilde{w} \in \widetilde{X}_e$$

where $c_e = \sum_{K_i} \frac{\text{area}(K_i)}{2}$ where $K_i$ are the elements such that $e \in \partial K_i$,

$$\begin{aligned} q_n &:= \frac{2}{(p+4+n)(p-n+1)} \int_{-1}^{1} (1-x^2)^2 P_n^{(2,2)}(x)^2 \, dx \\ &= \frac{64(n+1)(n+2)}{(p+4+n)(p-n+1)(2n+5)(n+3)(n+4)} \end{aligned} \tag{14}$$

and $\mu_n$ is the weighted moment given by

$$\mu_n(u) := \frac{(2n+5)(n+3)(n+4)}{32(n+1)(n+2)} \int_{-1}^{1} (1-x^2) P_n^{(2,2)}(x) u(x) \, dx$$

where we use a linear parametrization such that $e = [-1, 1]$.

Given $f \in X$, the additive Schwarz method from [5] is:

(i) $u_I \in X_I : a_I(u_I, v_I) = (f, v_I) \quad \forall v_I \in X_I$.

(ii) $u_V \in X_V : a_V(\widetilde{u}_V, \widetilde{v}_V) = (f, \widetilde{v}_V) \quad \forall \widetilde{v}_V \in \widetilde{X}_V$.

(iii) For all edges $e$ in $\mathcal{T}$, $\widetilde{u}_e \in \widetilde{X}_e : a_e(\widetilde{u}_e, \widetilde{v}_e) = (f, \widetilde{v}_e) \quad \forall \widetilde{v}_e \in \widetilde{X}_e$.

(iv) $u := u_I + \widetilde{u}_V + \sum_{e \in \mathcal{T}} \widetilde{u}_e$ is our solution.

The key result regarding the condition number is the following:

**Theorem 4.1.** *The condition number of the above additive Schwarz method is bounded by a constant $C$ independent of $h$ and $p$ (Theorem 3.1 from [5]).*

In the following sections, we discuss the implementation of each of the steps of the ASM preconditioner using a Bernstein basis. Let $\vec{B}_V^p, \vec{B}_E^p, \vec{B}_I^p$ be respectively the vectors such that its entries are the vertex, edge and interior Bernstein polynomials on the mesh. We enumerate the basis analogous to [5]:

1. the vertex functions $\vec{B}_V^p$ in any order

2. the edge functions grouped by edges, and ordered by the multi-indices

3. the interior functions grouped by the element which they are supported on

We can construct the mass matrix for the Bernstein basis on $\mathcal{T}$ in the following block form

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{VV} & \mathbf{M}_{VE} & \mathbf{M}_{VI} \\ \mathbf{M}_{EV} & \mathbf{M}_{EE} & \mathbf{M}_{EI} \\ \mathbf{M}_{IV} & \mathbf{M}_{IE} & \mathbf{M}_{II} \end{bmatrix}$$

where the subscripts indicate the interaction between vertices (V), edges (E) or interiors (I); the residual against the Bernstein polynomials $\vec{f}$ and solution $\vec{x}$ vector can be blocked in a similar way as

$$\bar{f} = \begin{bmatrix} \vec{f}_V \\ \vec{f}_E \\ \vec{f}_I \end{bmatrix} \text{ and } \bar{x} = \begin{bmatrix} \vec{x}_V \\ \vec{x}_E \\ \vec{x}_I \end{bmatrix}.$$

*4.3. Interior Spaces*

In this section, we give an efficient algorithm to solve

$$a_I(u, w) = (f, w) \qquad \forall w \in X_I. \tag{15}$$

As $X_I$ is the direct sum of bubble functions on each individual element, we can simply discuss the implementation on the reference element. For the sake of conciseness, we leave all proofs in this section to the appendix.

We recall an orthogonal basis for $\mathbb{P}^p(K) \cap H_0^1(K)$ is given by (see §2.1 of [5])

$$\psi_{ij}(x, y) = \frac{1-s}{2}\frac{1+s}{2}P_{i-1}^{(2,2)}(s)\left(\frac{1-t}{2}\right)^{i+1}\frac{1+t}{2}P_{j-1}^{(2i+3,2)}(t)$$

for $1 \le i, j, i+j \le p-1$, where

$$s = \frac{\lambda_2 - \lambda_1}{1-\lambda_3}, \quad t = 2\lambda_3 - 1$$

and $\lambda_1, \lambda_2, \lambda_3$ are the barycentric coordinates of $T$. If we let

$$u(x, y) = \sum_{i=1}^{p-1}\sum_{j=1}^{p-1-i} u_{ij}\psi_{ij}(x, y)$$

for coefficients $u_{ij}$, then plugging $u(x,y)$ into eq. (15) with the test functions $w = \psi_{lm}(x,y)$, we see that $u_{ij} = \frac{(f,\psi_{ij})}{\|\psi_{ij}\|^2}$, hence the solution to eq. (15) is simply

$$u(x,y) = \sum_{i=1}^{p-1} \sum_{j=1}^{p-1-i} \frac{(f,\psi_{ij})}{\|\psi_{ij}\|^2} \psi_{ij}(x,y) = \sum_{|\alpha|=p} c_\alpha^p B_\alpha^p. \tag{16}$$

Since we are working with the Bernstein polynomials, the question is now a matter of converting from the $\psi_{ij}$ basis to the Bernstein basis.

First, we rewrite the basis functions $\psi_{ij}$ as a multiple of $\lambda_1\lambda_2\lambda_3$, and make a change of variables on the indices obtaining

$$\psi_{ij}|_{r=i-1,\, m-r=j-1} = \lambda_1\,\lambda_2\,\lambda_3\, P_r^{(2,2)}(s) \left(\frac{1-t}{2}\right)^r P_{m-r}^{(2r+5,2)}(t),$$

for $0 \le r \le m$ and $0 \le m \le p-3$. The next lemma gives allows one to rewrite the interior basis functions as a sum of Bernstein polynomials.

**Lemma 4.2.** *Let $0 \le r \le m$ and $0 \le m \le p-3$. Then, it holds*

$$P_r^{(2,2)}(s) \left(\frac{1-t}{2}\right)^r P_{m-r}^{(2r+5,2)}(t) = \sum_{|\alpha|=m} a_\alpha^{mr} B_\alpha^m(x,y),$$

*where, for $|\alpha| = m$*

$$a_\alpha^{mr} = \begin{cases} \nu_{\alpha_3}^{mr} \gamma_{\alpha_2}^{r,m-\alpha_3}, & \text{for } \alpha_3 \le m-r, \\ 0, & \text{otherwise,} \end{cases}$$

*and*

$$\begin{aligned} \nu_{\alpha_3}^{mr} &= (-1)^{m-r-\alpha_3} \frac{\binom{m+r+5}{\alpha_3}\binom{m-r+2}{m-r-\alpha_3}}{\binom{m}{\alpha_3}}, \\ \gamma_{\alpha_2}^{r,m-\alpha_3} &= \sum_{l=0}^{m-r-\alpha_3} \gamma_{\alpha_2-l}^r \frac{\binom{m-r-\alpha_3}{l}\binom{r}{\alpha_2-l}}{\binom{m-\alpha_3}{\alpha_2}}, \\ \gamma_j^r &= (-1)^{r-j} \frac{\binom{r+2}{j}\binom{r+2}{r-j}}{\binom{r}{j}}, \end{aligned}$$

*for $j = 0, \ldots, r$. Note that $\gamma_j^r$ are the Bernstein-Bézier coefficients of the one-dimensional Jacobi polynomial $P_r^{(2,2)}$.*

To obtain the Bernstein-Bézier coefficients $c_\alpha^p$ of $u(x,y)$, we apply lemma 4.2 to eq. (16), obtaining

$$u(x,y) = \lambda_1\lambda_2\lambda_3 \sum_{m=0}^{p-3} \sum_{r=0}^{m} \sum_{|\alpha|=m} a_\alpha^m \frac{(f, \lambda_1\lambda_2\lambda_3 B_\alpha^m)}{\|\psi_{mr}\|^2} \sum_{|\beta|=m} a_\beta^{mr} B_\beta^m(x,y). \quad (17)$$

We remark that the form given as above is the sum of Bernstein polynomials of *different orders*; hence care must be taken to ensure that we express $u(x,y)$ as a sum of $p$th order Bernstein polynomials.

Considering that we are given the Bernstein moments $f_\alpha^p = (f, B_\alpha^p)$ of degree $p$ of a function $f$, we break down the calculations into 5 steps:

**Step 1.** Compute moments

$$\tilde{f}_\alpha^{p-3} = (f, \lambda_1\lambda_2\lambda_3 B_\alpha^{p-3}),$$

for the data $f^p$.

**Step 2.** Compute

$$S^{mr} = \sum_{|\alpha|=m} a_\alpha^{mr} \frac{\tilde{f}_\alpha^m}{\|\psi_{mr}\|^2}, \quad \text{for } r = 0,\ldots,m,\ m = 0,\ldots,p-3.$$

**Step 3.** Compute

$$T_\beta^m = \sum_{r=0}^{m} S^{mr} a_\beta^{mr}, \quad \text{for } |\beta| = m,\ m = 0,\ldots,p-3.$$

**Step 4.** Compute coefficients $c_\alpha^m$ by raising the coefficients $c_\alpha^{m-1}$ (if $m > 0$) to degree $m$ and adding them to $T^m$.

**Step 5.** Compute coefficients $c_\alpha^p$ from coefficients $c_\alpha^{p-3}$ by multiplying by the interior bubble function, i.e.

$$\sum_{|\alpha|=m} c_\alpha^p B_\alpha^p = \lambda_1\lambda_2\lambda_3 \sum_{|\alpha|=p-3} c_\alpha^{p-3} B_\alpha^{p-3}.$$

We will observe in the following sections that the costs of Steps 1, and 5 are of $\mathcal{O}(p^2)$, and Step 4 is of $\mathcal{O}(p^3)$. If we compute the sums in Steps 2 and 3 naively, we end up with a cost of $\mathcal{O}(p^4)$. This is of course not optimal in the

sense that it does not match the computational complexity of other algorithms in this paper. In the following subsections we present algorithms computing Steps 1-5, in particular the algorithms for Steps 2 and 3 use recurrence relations in the computations that allow us to achieve an optimal order. In summary, we obtain an algorithm for computing the Bernstein-Bézier coefficients $c_\alpha^p$ of $u(x,y)$ of computational complexity of $\mathcal{O}(p^3)$.

*4.3.1. Computation of Step 1.*

Since the residual vector $\vec{f_I} = (f, B_\alpha^p)$ for $\alpha$ the index set corresponding to the interior domain points, we then note that $\tilde{f}_\alpha^{p-3}$ is obtained by

$$(f, \lambda_1\lambda_2\,\lambda_3 B_\alpha^{p-3}) = \frac{(\alpha_1+1)(\alpha_2+1)(\alpha_3+1)}{(p-2)(p-1)p} f_{\alpha+(1,1,1)}^p.$$

for $|\alpha| = p-3$ as

$$\lambda_1\lambda_2\lambda_3 B_\alpha^{p-3} = \lambda_1\lambda_2\lambda_3 \frac{(p-3)!}{\alpha!}\lambda_1^{\alpha_1}\lambda_2^{\alpha_2}\lambda_3^{\alpha_3} = \frac{(\alpha_1+1)(\alpha_2+1)(\alpha_3+1)}{(p-2)(p-1)p} B_{\alpha+(1,1,1)}^p. \tag{18}$$

*4.3.2. Computation of Step 2*

We assume that $\tilde{f}_\alpha^{p-3}$ is given from Step 1, then we compute $S^{mr}$ for $r = 0,\ldots,m$ and $m = 0,\ldots,p-3$. Plugging in the definition of $a_\alpha^{mr}$, we have

$$\begin{aligned} S^{m,r} &= \frac{1}{\|\psi_{mr}\|^2}\sum_{|\alpha|=m} a_\alpha^{mr}\tilde{f}_\alpha^m = \frac{1}{\|\psi_{mr}\|^2}\sum_{\alpha_3=0}^{m-r}\nu_{\alpha_3}^{mr}\sum_{\alpha_1+\alpha_2=m-\alpha_3}\gamma_{\alpha_2}^{r,m-\alpha_3}\tilde{f}_\alpha^m \\ &:= \frac{1}{\|\psi_{mr}\|^2}\sum_{\alpha_3=0}^{m-r}\nu_{\alpha_3}^{mr}Q_{\alpha_3}^{mr}. \end{aligned}$$

The key to decreasing the number of operations is to use recurrence relations of the coefficients and moments. We claim that we can compute $Q_{\alpha_3}^{mr}$ and $S^{mr}$ with the following strategy:

1. Compute $Q_{\alpha_3}^{mr}$ and $S^{mr}$ for $\alpha_3 = 0,\ldots,m-r$, $r = 0,\ldots,m$ and $m = p-3$.

   This has a cost of $\mathcal{O}(p^3)$ as we have to iterate over $\alpha_3$ and $r$ for each $Q_{\alpha_3}^{mr}$, then compute a sum of $\mathcal{O}(p)$. We remark that while we can precompute and store the coefficients $\gamma_{\alpha_2}^{r,m-\alpha_3}$, we can compute these coefficients using a 1D degree raising algorithm (see lemma 7.1).

2. Compute recursively for $m = (p-3)-1, \ldots, 0$:

$$Q_{\alpha_3}^{mr} = \frac{\alpha_3 + 1}{m+1} Q_{\alpha_3+1}^{m+1r} + \frac{m+1-\alpha_3}{m+1} Q_{\alpha_3}^{m+1r}, \quad \alpha_3 = 0, \ldots, m-r,$$
$$S^{mr} = \frac{1}{\|\psi_{mr}\|^2} \sum_{\alpha_3=0}^{m} \nu_{\alpha_3}^{mr} Q_{\alpha_3}^{mr},$$

for $r = 0, \ldots, m$.

This step also has a cost of $\mathcal{O}(p^3)$ as we have to loop over $m, r, \alpha_3$ to calculate $Q_{\alpha_3}^{mr}$; $S^{mr}$ requires us to loop over $m, r$ and sum over $m$.

We prove this recurrence relation in the appendix.

*4.3.3. Computation of Step 3.*

Similar to Step 2, we compute the terms $T_\beta^m$ for each $m$ using the following strategy:

1. Compute $T_\beta^m$ for $|\beta| = m$ and $\beta_3 = 0$.

   Since we are looping over $|\beta| = m$ such that $\beta_3 = 0$, this loop is of $\mathcal{O}(p)$. We also have to calculate the sum for $T_\beta^m$ at each loop, which incurs a cost of $\mathcal{O}(p)$. Finally, we would have to do this for each $m$, meaning the final cost is of $\mathcal{O}(p^3)$.

2. Compute for $\beta_3 = 0, \ldots, m-1$

$$T_{\beta+e_3}^m = -\left( \frac{\beta_1 + 3}{\beta_3 + 3} T_{\beta+e_1}^m + \frac{\beta_2 + 3}{\beta_3 + 3} T_{\beta+e_2}^m \right),$$

   for $\beta_2 = 0, \ldots, m - \beta_3 - 1$.

The cost here is clearly $\mathcal{O}(p^3)$ as we need to loop over $m$, then over $\beta_2, \beta_3$.

*4.3.4. Computation of Step 4.*

We compute the coefficients $c^m$ by adding $T^m$ and $c_\alpha^{m-1}$. We observe that $c_\alpha^{m-1}$ corresponds to coefficients of degree $m-1$, then we use two dimensional degree raising to carry out the operation (algorithm 14). The 2D degree raise algorithm has a cost of $\mathcal{O}(p^2)$; we need to do this for all $m < p-3$, hence a cost of $\mathcal{O}(p^3)$.

*4.3.5. Computation of Step 5.*

Similarly to Step 1, we obtain the coefficients $c^p_\alpha$ by using eq. (18)

$$c^p_{\beta+1} = \frac{(\beta_1+1)(\beta_2+1)(\beta_3+1)}{(p-2)(p-1)p} c^{p-3}_\beta,$$

for $|\beta| = p-3$.

We outline the procedure for computing the coefficients in algorithms 4 and 5.

**Algorithm 4** Inversion of interior Bernstein mass matrix

**Require:** Interior Bernstein moments $f^p_\alpha = (f, B^p_\alpha)$, for $|\alpha| = p$, and $0 < \alpha < p$

**function** $\mathbf{M}_{II}^{-1}(f^p_\alpha)$
  **for** $|\alpha| = p-3$ **do** ▷ Step 1
    $\tilde{f}^{p-3}_\alpha = \frac{(\alpha_1+1)(\alpha_2+1)(\alpha_3+1)}{(p-2)(p-1)p} f^p_{\alpha+1}$
  **end for**
  $c^{p-3}_\alpha =$ `InteriorInverse` $(\tilde{f}^{p-3}_\alpha)$
  **for** $|\alpha| = p-3$ **do** ▷ Step 5
    $c^p_{\alpha+1} = \frac{(\alpha_1+1)(\alpha_2+1)(\alpha_3+1)}{(p-2)(p-1)p} c^{p-3}_\alpha$
  **end for**
  **return** $c^p_\alpha$
**end function**

**Algorithm 5** Computing interior coefficients

**function** InteriorInverse($\tilde{f}_\alpha^{p-3}$)
  $n = p - 3$ ▷ For convenience
  **for** $r = 0, \ldots, n$ **do** ▷ Step 2: Initialize $Q^{nr}$ and $S^{nr}$
    $\gamma^{r,r}$ = Jacobi (2,2)-Bernstein coefficients of degree $r$
    $\alpha_3 = n - r$
    $Q^{n,r}_{\alpha_3} = \gamma^{r,r} \cdot \tilde{f}^n_{\cdot,\alpha_3}$
    **for** $\alpha_3 = n - r - 1, \ldots, 0$ **do**
      $\gamma^{r,n-\alpha_3} = \texttt{DegreeRaise}(\gamma^{r,n-\alpha_3-1})$
      $Q^{n,r}_{\alpha_3} = \gamma^{r,n-\alpha_3} \cdot \tilde{f}^n_{\cdot,\alpha_3}$
    **end for**
    $S^{n,r} = \sum_{\alpha_3=0}^{n-r} \nu^{n,r}_{\alpha_3} Q^{n,r}_{\alpha_3} / \|\psi_{nr}\|^2$
  **end for**
  **for** $m = n - 1, \ldots, 0$ **do** ▷ Step 2: Recursive portion
    **for** $r = 0, \ldots, m$ **do**
      **for** $\alpha_3 = 0, \ldots, m - r$ **do**
        $Q^{m,r}_{\alpha_3} = \frac{(\alpha_3+1)}{m+1} Q^{m+1,r}_{\alpha_3+1} + \frac{(m+1-\alpha_3)}{(m+1)} Q^{m+1,r}_{\alpha_3}$
      **end for**
      $S^{m,r} = \sum_{\alpha_3=0}^{m-r} \nu^{m,r}_{\alpha_3} Q^{m,r}_{\alpha_3} / \|\psi_{nr}\|^2$
    **end for**
  **end for**
  **for** $m = 0, n$ **do**
    $T^m_{\cdot,0} = \sum_{r=0}^{m} S^{m,r} \nu^{m,r}_0 \gamma^{r,m}$ ▷ Step 3: Initialize
    **for** $\beta_3 = 0, m - 1$ **do**
      **for** $\beta_1 = 0, m - \beta_3 - 1$ **do**
        $\beta_2 = m - (\beta_3 + 1) - \beta_1$ ▷ Step 3: Recursive portion
        $T^m_{\beta+e_3} = -\frac{(3+\beta_1)}{(3+\beta_3)} T^m_{\beta+e_1} - \frac{(3+\beta_2)}{(3+\beta_3)} T^m_{\beta+e_2}$
      **end for**
    **end for**
    $c^m += T^m$ ▷ Step 4
    **if** $m < n$ **then**
      $c^{m+1} = \texttt{DegreeRaise2D}(c^m)$
    **end if**
  **end for**
  **return** $c_\alpha^{p-3}$
**end function**

*4.4. Edge Spaces*

In this section, we give efficient algorithms to solve $a_e(\widetilde{u},\widetilde{v}) = (f,\widetilde{v})$ for all $\widetilde{v} \in \widetilde{X}_e$ for a given edge $e$. Without loss of generality, we assume that $e = [-1,1]$ and that $c_e = 1$. The key to the efficient solver on the edge space is to note that the bilinear form $a_e$ only depends on the value restricted to $e$ which allows us to reformulate the problem over the space $X_e$ rather than $\widetilde{X}_e$.

We break down the edge solver into four distinct steps

**Step 1.** Reduce the variational form to $X_e$

**Step 2.** Compute the residual

**Step 3.** Compute $a_e(\vec{\varphi},\vec{\varphi})$ by using a change of basis

**Step 4.** Find the corresponding solution in $\widetilde{X}_e$.

*4.4.1. Step 1.*

Decompose $\widetilde{u}_e, \widetilde{v}_e \in \widetilde{X}_e$ into edge functions and bubble functions

$$\widetilde{u}_e = u_e + u_b \qquad\qquad \widetilde{v}_e = v_e + v_b$$

where $u_e, v_e \in X_e$, $u_b, v_b \in X_I$ and $(u_b, w_I) = -(u_e, w_I)$ for all $w_I \in X_I$, and analogously for $v_b$.

On the right hand side

$$\begin{aligned}(f,\widetilde{v}_e) &= (f,v_e) + (f,v_b)\\ &= (f,v_e) + (u_I,v_b)\\ &= (f,v_e) - (u_I,v_e) + (u_I,\widetilde{v}_e) = (f,v_e) - (u_I,v_e).\end{aligned}$$

where $u_I$ is the solution to $(u_I,w_I) = (r,w_I)$ for all $w_I \in X_I$ (i.e. the solution to the interior problem in section 4.3). We also used the fact that $(w_I,\widetilde{v}_e) = 0$ for all $w_I \in X_I$ by definition of $\widetilde{X}_e$.

For the bilinear form, we note that

$$a_e(\widetilde{u}_e,\widetilde{v}_e) = a_e(u_e,v_e) = a_e(u_e|_e, v_e|_e) \qquad \forall \widetilde{v}_e \in \widetilde{X}_e$$

where $u_e|_e$ is the restriction onto $e$. Hence, we can first find $u_e \in X_e$ such that

$$a_e(u_e, v_e) = (f, v_e) - (u_I, v_e) \qquad \forall v_e \in X_e, \tag{19}$$

then use the orthogonality property to find $\widetilde{u}_e \in \widetilde{X}_e$.

*4.4.2. Computation of Step 2.*

Let $\vec{B}_e^p$ be the Bernstein edge polynomials corresponding to the domain points on $e$ (i.e. a basis for $X_e$), then we see that the right hand side of eq. (19) is

$$(f, \vec{B}_e^p) - (u_I, \vec{B}_e^p) = \vec{f}_e - \mathbf{M}_{eI}\vec{u}_I$$

where $\mathbf{M}_{eI}$ the the mass matrix block corresponding to the interaction between $\vec{B}_e^p$ and the interior Bernstein basis, and $\vec{u}_I$ is vector B-form of the solution to $(u_I, w_I) = (f, w_I)$ for all $w_I \in X_I$ (section 4.3). We incur a cost of $\mathcal{O}(p^3)$ here due to the matrix multiply of $\mathbf{M}_{eI}\vec{u}_I$.

*4.4.3. Computation of Step 3.*

Let

$$\vec{\varphi}_e = \begin{bmatrix} (1-x^2)P_0^{(2,2)} \\ \vdots \\ (1-x^2)P_{p-2}^{(2,2)} \end{bmatrix};$$

$\vec{\varphi}_e$ spans the same space as the univariate "interior" Bernstein polynomials (i.e. space spanned by $\{B_1^p, \ldots, B_{p-1}^p\}$). Due to orthogonality of Jacobi polynomials,

$$a_e(\vec{\varphi}_e, \vec{\varphi}_e^T) := \mathbf{D}_{ee} = \mathrm{diag}(q_n)$$

for $0 \le n \le p-2$ where $q_n$ is from eq. (14) (see §5.1 of [5]).

We use the crucial fact that bivariate Bernstein polynomials restricted to the boundary are simply the univariate Bernstein polynomials [18]; hence $X_e$ restricted to $e$ is simply the span of univariate interior Bernstein polynomials

$\{B_1^p, B_2^p, \ldots, B_{p-1}^p\}$. Let us introduce the change of basis matrix $\mathbf{\Gamma}_e$ such that $\vec{\varphi}_e = \mathbf{\Gamma}_e(\vec{B}_e|_e)$ on $e$, then

$$\mathbf{D}_{ee} = a_e(\vec{\varphi}_e, \vec{\varphi}_e^T) = a_e(\mathbf{\Gamma}_e(\vec{B}_e|_e), (\mathbf{\Gamma}_e(\vec{B}_e|_e))^T) = \mathbf{\Gamma}_e a_e(\vec{B}_e, \vec{B}_e^T)\mathbf{\Gamma}_e^T.$$

The bilinear form under the Bernstein basis $a_e(\vec{B}_e, \vec{B}_e)$ corresponds to $\mathbf{\Gamma}_e^{-1}\mathbf{D}_{ee}\mathbf{\Gamma}_e^{-T}$. In order to invert this, we need efficient ways to compute $\mathbf{\Gamma}_e$ and $\mathbf{\Gamma}_e^T$.

Rather than store the matrix $\mathbf{\Gamma}_e$, we present algorithms which can compute their actions. For the action of $\mathbf{\Gamma}_e$, we note that given a function $f$ on $e$

$$\vec{\varphi}_e = \mathbf{\Gamma}_e(\vec{B}_e|_e) \implies (f, \vec{\varphi}_e) = \mathbf{\Gamma}_e(f, (\vec{B}_e|_e)),$$

hence the operator $\mathbf{\Gamma}_e$ converts the residual from the 1D Bernstein basis to the residual with respect to $\vec{\varphi}_e$ basis:

$$\mathbf{\Gamma}_e : \begin{bmatrix} (f, B_1^p) \\ \vdots \\ (f, B_{p-1}^p) \end{bmatrix} \to \begin{bmatrix} (f, (1-x^2)P_0^{(2,2)}) \\ \vdots \\ (f, (1-x^2)P_{p-2}^{(2,2)}) \end{bmatrix}$$

and likewise $\mathbf{\Gamma}_e^T$ is the operator which converts the coefficients of a polynomial expanded with $\vec{\varphi}_e$ into the B-form coefficients as follows:

$$\mathbf{\Gamma}_e^T : (1-x^2)\sum_{j=0}^{p-2} w_j P_j^{(2,2)}(x) \to \sum_{j=1}^{p-1} c_j B_j^p(x).$$

The key identity to an efficient implementation of $\mathbf{\Gamma}_e$ is

$$(1-x^2)P_n^{(2,2)} = 4\sum_{i=0}^{n} \frac{\binom{n+2}{i}\binom{n+2}{n-i}}{(-1)^{n-i}\binom{n}{i}} \frac{i+1}{n+1}\frac{n-i+1}{n+2} B_{i+1}^{n+2}, \tag{20}$$

which is obtained from eq. (12) and eq. (25), hence

$$(f, (1-x^2)P_n^{(2,2)}) = 4\sum_{i=0}^{n} \frac{\binom{n+2}{i}\binom{n+2}{n-i}}{(-1)^{n-i}\binom{n}{i}} \frac{i+1}{n+1}\frac{n-i+1}{n+2} (f, B_{i+1}^{n+2}).$$

Thus, knowing $(f, B_1^p), \ldots, (f, B_{p-1}^p)$ allows us to calculate $(f, (1-x^2)P_{p-2}^{(2,2)})$. A degree lowering operation (see algorithm 13) can then be used to obtain $(f, B_1^{p-1}), \ldots, (f, B_{p-2}^{p-1})$ which allows us to calculate $(f, (1-x^2)P_{p-3}^{(2,2)})$. We can

recursively do this to figure out the rest of the residuals. The following function performs $\mathbf{\Gamma}_e$:

**Algorithm 6** $\Gamma_e$: Converts $(f, B_i^p)$ into $(f, (1-x^2)P_i^{(2,2)})$

**Require:** $\vec{b}$, a vector of length $p-1$

1: **function** GAMMA(b)
2: **for** $i = p-2$ to $0$ **do**
3: **for** $j = 0, \ldots, i$ **do**
4: $o[i] = o[i] + 4\frac{\binom{i+2}{j}\binom{i+2}{i-j}}{(-1)^{i-j}\binom{i}{j}}\frac{j+1}{i+1}\frac{i-j+1}{i+2}b[j]$
5: **end for**
6: $\vec{b} := \texttt{DegreeLower}(\vec{b})$ ▷ Degree lower moments; cost of $\mathcal{O}(p)$
7: **end for**
8: **return** $\vec{o}$
9: **end function**

We note that `Gamma` clearly has a cost of $\mathcal{O}(p^2)$.[3]

The key to computing $\mathbf{\Gamma}_e^T$ is to use eq. (20) again. Starting with $w_0$, we can find the coefficients with respect to $B_1^2$. We perform a degree raising operation on the $B_1^2$ coefficient to obtain the coefficients in $B_1^3, B_2^3$. Now, we can use $w_1$ to find the coefficient with respect to $B_1^3, B_2^3$ and sum. We keep on degree raising, and using eq. (20) to obtain the following algorithm for $\mathbf{\Gamma}_e^T$:

[3]The coefficients can be computed with little cost by noting that

$$4\frac{\binom{i+2}{j}\binom{i+2}{i-j}}{(-1)^{i-j}\binom{i}{j}}\frac{j+1}{i+1}\frac{i-j+1}{i+2} = 4\binom{i+2}{j}\frac{i-j+1}{(j+2)(-1)^{i-j}}$$

hence one can either pre-computing the binomial coefficients up to order $p$, or updating the binomial coefficients in the for loop for $i$ on the fly while calculating `Gamma`

**Algorithm 7** $\mathbf{\Gamma}_e^T$: Converts the coefficients of a Jacobi polynomial to a B-form coefficients

**Require:** $w$, a vector of length $p-1$

**function** GAMMATRAN(w)
  Initialize $o$ of length 1
  **for** $i = 0, \ldots, p-1$ **do**
    **for** $j = 0, \ldots, i$ **do**
      $o[i] = o[i] + 4\frac{\binom{i+2}{j}\binom{i+2}{i-j}}{(-1)^{i-j}\binom{i}{j}}\frac{j+1}{i+1}\frac{i-j+1}{i+2}w[j]$
    **end for**
    $o = \texttt{DegreeRaise}(o)$ ▷ Degree raise B-net; cost of $\mathcal{O}(p)$
  **end for**
  **return** $o$
**end function**

Again, we see that `Gammatran` also have a cost of $\mathcal{O}(p^2)$ as the coefficients can be calculated as before. Hence, we can easily compute $\mathbf{\Gamma}_e^T\mathbf{D}_{ee}^{-1}\mathbf{\Gamma}_e$ with cost of $\mathcal{O}(p^2)$ and efficiently compute the solution to the variational problem eq. (19)

$$\vec{u}_e := \mathbf{\Gamma}_e^T\mathbf{D}_{ee}^{-1}\mathbf{\Gamma}_e(\vec{f_e} - \mathbf{M}_{eI}\vec{u}_I).$$

*4.4.4. Step 4.*

Finally, we recall $\vec{u}_e$ from above corresponds to

$$u_e \in X_e : a_e(u_e, v_e) = (f, v_e) - (u_I, v_e) \qquad \forall v_e \in X_e$$

but the solution we need is $\widetilde{u}_e$ in $\widetilde{X}_e$ with $\widetilde{u}_e = u_e + u_b$. Recall that $(u_b, w_I) = -(u_e, w_I)$ for all $w_I \in X_I$, hence the interior correction can be computed by

$$\vec{u}_b = -\mathbf{M}_{II}^{-1}\mathbf{M}_{Ie}\vec{u}_e.$$

This has a cost of $\mathcal{O}(p^3)$ if we use algorithm 4.

*4.5. Vertex Spaces*

In this section, we discus how to solve the variational problem $a_V(\widetilde{u}_v, \widetilde{w}_v) = (f, \widetilde{w}_v)$ for $\forall \widetilde{w}_v \in \widetilde{X}_V$. We proceed similarly to the edge solves.

**Step 1.** Reduce the variational form to $X_V$

**Step 2.** Perform change of basis, and calculate the residual

**Step 3.** Compute the bilinear form

**Step 4.** Find the corresponding solution in $\widetilde{X}_V$

*4.5.1. Step 1.*

Decompose $\widetilde{u}_v = u_v + u_b$ where $u_v \in X_V, u_b \in X_I$ with a similar decomposition for the test function $\widetilde{w}_v$. By the orthogonality property of $\widetilde{X}_V$, we have that

$$(u_v, w_b) = -(u_b, w_b) \qquad \forall w_b \in X_I. \tag{21}$$

and we also recall that

$$(u_I, w_b) = (f, w_b) \qquad \forall w_b \in X_I. \tag{22}$$

For the right hand side, we have again

$$\begin{aligned}(f, \widetilde{w}_v) &= (f, w_v) + (f, w_b) \\ &= (f, w_v) + (u_I, \widetilde{w}_v) - (u_I, w_v) \\ &= (f, w_v) - (u_I, w_v).\end{aligned}$$

As $a_V(\widetilde{u}_v, \widetilde{w}_v) = a_V(u_v, w_v)$, we can find $u_v \in X_V$

$$a_V(u_v, w_v) = (f, w_v) - (u_I, w_v) \qquad \forall w_v \in X_V \tag{23}$$

then use orthogonality properties to find $\widetilde{u}_v \in \widetilde{X}_V$.

*4.5.2. Step 2.*

Unfortunately, $X_V$ is not simply the span of the Bernstein polynomials. For an arbitrary vertex $v \in \mathcal{T}$, we note that we can rewrite the basis function $\varphi_v$ as a linear combination of Bernstein polynomials

$$\varphi_v = B_v^p + \vec{\phi}_v^T \vec{B}_E^p + \vec{\chi}_v^T \vec{B}_I^p \tag{24}$$

where $B_v^p$ is the Bernstein vertex basis at vertex $v$, $\vec{\phi}_v$ and $\vec{\chi}_v$ are vectors of appropriate coefficients. We will see that we need to compute $\vec{\phi}_v$, but not $\vec{\chi}_v$.

On the right hand side, using eq. (22) for an arbitrary vertex $v \in \mathcal{T}$ and the fact that $(u_I, w_I) = (f, w_I)$ for all $w_I \in X_I$,

$$\begin{aligned}(f, \varphi_v) - (u_I, \varphi_v) &= (f, B_v^p + \vec{\phi}_v^T \vec{B}_E^p + \vec{\chi}_v^T \vec{B}_I^p) - (u_I, B_v^p + \vec{\phi}_v^T \vec{B}_E^p + \vec{\chi}_v^T \vec{B}_I^p) \\ &= (f, B_v^p) - (u_I, B_v^p) + (f, \vec{\phi}_v^T \vec{B}_E^p) - (u_I, \vec{\phi}_v^T \vec{B}_E^p) \\ &= (\vec{f}_V)_v - (\mathbf{M}_{VI} u_I)_v + \vec{\phi}_v^T (\vec{f}_E - \mathbf{M}_{EI} u_I)\end{aligned}$$

where $(\vec{f}_V)_v$ and $(\mathbf{M}_{VI} u_I)_v$ is the row corresponding to $B_v^p$. The key here is that the interior component does not matter in the computation.

We need to compute $\vec{\phi}_v^T$ for all vertices $v$ which are the coefficients such that $B_v^p + \vec{\phi}_v^T B_E^p$ on the edges equals $\varphi_v$. Without loss of generality, given a vertex $v$, assume an edge $e$ from $v$ is parametrized to be $[-1, 1]$. We recall that $\varphi_v$ restricted to the edge is eq. (13), hence using eq. (12), and factoring in the $(1 - x)/2$ term,

$$\begin{aligned}\varphi_v(x)|_e &= \frac{(-1)^{\lfloor p/2 \rfloor + 1}}{\lfloor p/2 \rfloor} \left( \frac{1 - x}{2} \right) P_{\lfloor p/2 \rfloor - 1}^{(1,1)}(x) \\ &= \frac{1}{\lfloor p/2 \rfloor} \left( \sum_{j=0}^{\lfloor p/2 \rfloor - 1} \binom{\lfloor p/2 \rfloor}{\lfloor p/2 \rfloor - 1 - j} (-1)^j B_j^{\lfloor p/2 \rfloor}(x) \right) \\ &= B_0^p(x) + \sum_{j=1}^{p-1} \widetilde{c}_j B_j^p(x).\end{aligned}$$

Hence the coefficients we want are $\widetilde{c}_j$, which are the result of using the degree raising formula on $\binom{\lfloor p/2 \rfloor}{\lfloor p/2 \rfloor - 1 - j}(-1)^j$.

Let $\boldsymbol{\phi}$ be the matrix with columns the vector $\vec{\phi}_v$; algorithm 8 calculates $\boldsymbol{\phi}$ by first computing the coefficients for $B_j^{\lfloor p/2 \rfloor}$, degree raising it to the appropriate order $B_j^p$, then place the coefficients in the appropriate degrees of freedom in $\boldsymbol{\phi}$. We note that in line 9, we remove the first and last term as that corresponds to the vertex terms. $\boldsymbol{\phi}$ can be precomputed.

**Algorithm 8** Computing the values of $\boldsymbol{\phi}$

$\boldsymbol{\phi} = \text{zeros}(\text{numbers of dofs on edges}, \text{number of vertices})$ ▷ Initialize Matrix
$q := \lfloor p/2 \rfloor$
**for** $i = 0$ to $q$ **do**
    $\vec{c}[i] = \frac{(-1.0)^i}{q}\binom{q}{q-1-i}$ ▷ Generate lower-order coefficients
**end for**
**for** $i = 0, \ldots, p-q-1$ **do**
    $\vec{c} = \texttt{DegreeRaise1D}(\vec{c})$
**end for**
$\vec{c} = \vec{c}[1 : p-1]$ ▷ Remove first and last term; length of $p-1$
**for** $K \in \mathcal{T}$ **do**
    **for** vertex $v_i$ in $K$ **do**
        Let $v_j, v_k$ be the two other vertices of $K$
        $\vec{d_1} :=$ DOFs on edge from vertex $v_i$ to $v_j$
        $\vec{d_2} :=$ DOFs on edge from vertex $v_i$ to $v_k$
        $\boldsymbol{\phi}[\vec{d_1}, \text{dof of } v_i] = \vec{c}$ ▷ Set an array equal to another array
        $\boldsymbol{\phi}[\vec{d_2}, \text{dof of } v_i] = \vec{c}$
    **end for**
**end for**

*4.5.3. Step 3.*

The matrix form of the bilinear form is trivial as

$$a_V(\widetilde{u}_v, \widetilde{w}_v) = a_V(u_v, w_v) = \frac{1}{p^4}\mathbf{c}_v$$

where $\mathbf{c}_v$ is a diagonal matrix with $c_v$ as its entries. With the matrix $\boldsymbol{\phi}$ computed, we can solve for eq. (23) with the following

$$\frac{1}{p^4}\mathbf{c}_v\vec{\varphi}_v = \vec{f}_V - \mathbf{M}_{VI}\vec{u}_I + \boldsymbol{\phi}^T(\vec{f}_E - \mathbf{M}_{EI}\vec{u}_I).$$

The cost to compute $\vec{\varphi}_v$ is dependent on the number of vertices, but the main cost is $\mathcal{O}(p^3)$ due to the matrix-vector multiply of $\mathbf{M}_{EI}\vec{u}_I$.

*4.5.4. Step 4.*

Finally, the solution vector $\vec{\varphi}_v$ is under the $\varphi_v$ basis of $X_V$ so we have to manipulate this solution in order to find the corresponding solution in $\widetilde{X}_V$ expanded with the Bernstein basis.

Using eq. (24), the coefficient $\vec{u}_v$ for $B_V^p$ is simply $\vec{\varphi}_v$, and the coefficients for the edge Bernstein polynomials are $\vec{u}_E = \boldsymbol{\phi}\vec{\varphi}_v$.

As for the interior bubble functions, we recall the orthogonality condition eq. (21). Hence, we do not need to compute $\vec{\chi}_v^T$, but only the following variational problem for all $w_b \in X_I$

$$(\vec{u}_v^T B_V^p + \vec{u}_E^T B_E^p, w_b) = -(\vec{u}_b^T B_I^p, w_b) \implies \mathbf{M}_{IV}\vec{u}_v + \mathbf{M}_{IE}\vec{u}_E = -\mathbf{M}_{II}\vec{u}_b$$

and and hence $\vec{u}_b = -\mathbf{M}_{II}^{-1}\mathbf{M}_{IV}\vec{\varphi}_v - \mathbf{M}_{II}^{-1}\mathbf{M}_{IE}\boldsymbol{\phi}\vec{\varphi}_v$. The cost to compute this is $\mathcal{O}(p^3)$ if we use algorithm 4.

*4.6. Matrix Formulation*

Collecting the algorithms above, we can finally display the algorithm to precondition the mass matrix of the Bernstein basis. Let the local assembly matrix $\mathbf{\Lambda}_K$ be written in block form

$$\mathbf{\Lambda}_K = \begin{bmatrix} \mathbf{\Lambda}_{K,V} \\ \mathbf{\Lambda}_{K,E} \\ \mathbf{\Lambda}_{K,I} \end{bmatrix}$$

where the blocks correspond to the vertex, edge and interior basis functions on element $K$, then let the matrices $\mathbf{D}_{EE}$ and $\mathbf{D}_{VV}$ be diagonal matrices defined as

$$\mathbf{D}_{VV} = \sum_{K\in\mathcal{T}} \frac{|K|}{2p^4}\mathbf{\Lambda}_{K,V}\mathbf{\Lambda}_{K,V}^T \text{ and } \mathbf{D}_{EE} = \sum_{K\in\mathcal{T}} \frac{|K|}{2}\mathbf{\Lambda}_{K,E}\hat{\mathbf{D}}_{EE}\mathbf{\Lambda}_{K,E}^T$$

where

$$\hat{\mathbf{D}}_{EE} = \text{block diag}(\hat{\mathbf{D}}_{EE}^{(1)}, \hat{\mathbf{D}}_{EE}^{(2)}, \hat{\mathbf{D}}_{EE}^{(3)})$$

for $\hat{\mathbf{D}}_{EE}^{(i)}, i = 1, 2, 3$ is the diagonal matrix $\hat{\mathbf{D}}_{EE}^{(i)} = \text{diag}(q_j)$, with $q_j$ defined from eq. (14) for $j = 0, \ldots, p-2$. We let $\mathbf{\Gamma}_{EE}$ and $\mathbf{\Gamma}_{EE}^T$ simply be the applications of algorithms $\mathbf{\Gamma}_e, \mathbf{\Gamma}_e^T$ repeatedly for each edge.

Then, we can formulate the additive Schwarz preconditioner as

**Algorithm 9 P**: Preconditioner for the Bernstein Basis Mass Matrix

**Require:** $\mathbf{M}$ global mass matrix, $\vec{f}$ residual vector

**function**

$\vec{x}_I := \mathbf{M}_{II}^{-1} \vec{f}_I$ ▷ Interior solve using section 4.3

$\vec{x}_E := \mathbf{\Gamma}_{EE}^{-T} \mathbf{D}_{EE}^{-1} \mathbf{\Gamma}_{EE}^{-1} \left( \vec{f}_E - \mathbf{M}_{EI} \vec{x}_I \right)$ ▷ Edges solve

$\vec{x}_V := \mathbf{D}_{VV}^{-1} \left( (\vec{f}_V - \mathbf{M}_{VI} \vec{x}_I) + \boldsymbol{\phi}^T \left( \vec{f}_E - \mathbf{M}_{EI} \vec{x}_I \right) \right)$ ▷ Vertices solve

$\vec{x}_E := \vec{x}_E + \boldsymbol{\phi} \vec{x}_V$

$\vec{x}_I := \vec{x}_I - \mathbf{M}_{II}^{-1} \mathbf{M}_{IV} \vec{x}_V - \mathbf{M}_{II}^{-1} \mathbf{M}_{IE} \vec{x}_E$ ▷ Interior correction

**return** $x := [\vec{x}_V; \vec{x}_E; \vec{x}_I]$

**end function**

### *4.7. Schur Preconditioner*

In this subsection, we present a variation of the above algorithm which is more suited for explicit time-stepping such as the Nyström method or an explicit Runge-Kutta method. We first let $\mathbf{M}$ be the global mass matrix with the Bernstein basis, and block the matrix as follows

$$\mathbf{A} = \begin{bmatrix} \mathbf{M}_{VV} & \mathbf{M}_{VE} \\ \mathbf{M}_{EV} & \mathbf{M}_{EE} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{M}_{VI} \\ \mathbf{M}_{EI} \end{bmatrix}, \text{ and } \mathbf{C} = \begin{bmatrix} \mathbf{M}_{II} \end{bmatrix}.$$

One way of solving $\mathbf{M}\vec{x} = \vec{f}$ is to use the Schur complement method (otherwise known as static condensation [49]) by first solving the boundary values:

$$\ddot{\mathbf{S}} \begin{bmatrix} \vec{x}_V \\ \vec{x}_E \end{bmatrix} = \begin{bmatrix} \vec{f}_V \\ \vec{f}_E \end{bmatrix} - \mathbf{B}\mathbf{C}^{-1} \vec{f}_I = \begin{bmatrix} \vec{f}_V - \mathbf{M}_{VI} \mathbf{M}_{II}^{-1} \vec{f}_I \\ \vec{f}_E - \mathbf{M}_{EI} \mathbf{M}_{II}^{-1} \vec{f}_I \end{bmatrix}$$

where $\ddot{\mathbf{S}} = \mathbf{A} - \mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T$, then substitute the solution back to solve for the interior $\vec{x}_I$.

In the case of an explicit time-stepping scheme, we are able to solve for the right-hand side (e.g. $\vec{f}_V - \mathbf{M}_{VI}\mathbf{M}_{II}^{-1}\vec{f}_I$) exactly using section 4.3 and work with the exact Schur complement of the mass matrix.[4] Hence, rather than using conjugate gradient over the mass matrix, *we can simply iterate (and precondition)* on the smaller Schur complement then substitute back into the interior dofs. This idea was first mentioned in Remark 2.7 of [10].

The Schur complement preconditioner is the "middle" portion of algorithm 9 and is presented in algorithm 10 independently. Here, we again emphasize that using the Bernstein basis allows for matrix-free computation of the matrix-vector product [2], which coupled with the inversion of the interior blocks section 4.3 allows for matrix-free Schur complement products. Finally, the preconditioner for the whole mass matrix based on preconditioning the Schur complement is presented in algorithm 11.

**Algorithm 10** $\ddot{\mathbf{P}}^{-1}$: Preconditioner for Schur Complement

**Require:** $\vec{f}$ residual vector

1: **function**
2: $\quad \vec{x}_E := \mathbf{\Gamma}_{EE}^{-T}\mathbf{D}_{EE}^{-1}\mathbf{\Gamma}_{EE}^{-1}\left(\vec{f}_E\right)$ ▷ Edges solve
3: $\quad \vec{x}_V := \mathbf{D}_{VV}^{-1}\left(\vec{f}_V + \boldsymbol{\phi}\vec{f}_E\right)$ ▷ Vertices solve
4: $\quad \vec{x}_E := \vec{x}_E + \boldsymbol{\phi}\vec{x}_V$
5: $\quad$ **return** $x := [x_V; x_E]$
6: **end function**

[4] This is contrasted against an implicit time-stepping scheme where the right hand side in the Schur complement method will requires $(\mathbf{M}_{II} + c\mathbf{S}_{II})^{-1}$ which cannot be as easily computed exactly as $\mathbf{M}_{II}$.

**Algorithm 11** $\tilde{\mathbf{P}}$: Preconditioner for Mass Matrix using $\ddot{\mathbf{P}}^{-1}$

**Require:** $\mathbf{M}$ global Bernstein mass matrix, $\ddot{\mathbf{S}}$ Schur complement of Bernstein basis, $\vec{f}$ residual vector

1: **function**
2: $\vec{x}_I := \mathbf{M}_{II}^{-1}\vec{f}_I$ ▷ Interior solve
3: $\widetilde{f}_V = \vec{f}_V - \mathbf{M}_{VI}\mathbf{M}_{II}^{-1}\bar{f}_I$ ▷ Find right-hand sides for Schur complement
4: $\widetilde{f}_E = \vec{f}_E - \mathbf{M}_{EI}\mathbf{M}_{II}^{-1}\bar{f}_I$
5: $[\vec{x}_V; \vec{x}_E] := \texttt{pcg}(\ddot{\mathbf{S}}, [\widetilde{f}_V; \widetilde{f}_E], \text{Preconditioner} = \ddot{\mathbf{P}}^{-1})$ ▷ Iterate the boundaries
6: $\vec{x}_I := \vec{x}_I - \mathbf{M}_{II}^{-1}\mathbf{M}_{IV}\vec{x}_V - \mathbf{M}_{II}^{-1}\mathbf{M}_{IE}\vec{x}_E$ ▷ Interior correction
7: **return** $x := x_I + x_E + x_V$
8: **end function**

## 5. Illustrative Numerical Examples

### *5.1. Brusselator and Implicit Time-Stepping*

We now illustrate the use of the preconditioner in the numerical solution of the Brusselator system. Let $u(x, y, t)$ and $v(x, y, t)$ be the solution to the Brusselator system with initial conditions and time-stepping scheme as described in section 2.2. The spatial discretization is a uniform triangulation of the square with 256 elements.

In table 2, we show the [min, median, max] of the iteration counts of the preconditioned conjugate gradient (PCG) method required to solve for both $u(x, y, t)$ and $v(x, y, t)$ separately. We note that while we are preconditioning a perturbation of the mass matrix, the choice of $\Delta t \sim \frac{h^2}{p^2}$ and a good initial iterate seems to allow us to have *non-increasing* iteration counts as opposed to the $\mathcal{O}(p^2)$ growth shown in section 2.2. This is partly due to the fact that the diffusion coefficient is so small, and the fact that we are using the previous time-step as the initial iterate for PCG.

We also will use this case study to showcase the advantages of using the Bernstein basis in calculating the critical nonlinear moments at each time-step as

mentioned in section 3.5. In fig. 12, we plot the average number of milliseconds required to calculate the nonlinear moment $(u_n^2 v_n, \vec{\varphi})$ at each time step.[5] We note that while [2] indicated that the asymptotic cost is $\mathcal{O}(p^3)$, in the range of $p \in [3, \ldots, 20]$, we instead see a better cost growth of only $\mathcal{O}(p^2)$.

Table 2: Table to illustrate the performance of the preconditioned iterative method to the matrix resulting from a IMEX scheme by displaying the [min, median, max] iteration count of the PCG solves for the variable $u$ and $v$ in a period of 10 seconds on 256 elements for the Brusselator in a uniformly triangulated square. Our scaling for $\Delta t$ is such that $\Delta t \sim \frac{1}{p^2}$.

| $p$ | $\Delta t$ | Iteration count $u$ | Iteration count $v$ |
|---|---|---|---|
| 4 | 1/10 | [22, 25, 28] | [24, 27, 31] |
| 8 | 1/40 | [19, 22, 26] | [20, 23, 27] |
| 12 | 1/90 | [18, 21, 24] | [20, 22, 26] |
| 16 | 1/160 | [18, 21, 26] | [19, 23, 26] |
| 20 | 1/250 | [18, 22, 27] | [20, 23, 27] |

### *5.2. Sine-Gordon and Explicit Time-Stepping*

We now illustrate the use of the preconditioner in the numerical solution of the sine-Gordon equation, using an explicit time-stepping scheme which requires the inversion of the exact mass matrix at each step. Let $u(x, y, t)$ be the solution to the sine-Gordon equation using the fourth order Nyström method [25, p. 285] as described in section 2.1. We use a uniform triangulation of the square $[-7, 7] \times [-7, 7]$ in the spatial dimension.

In order to time-step, we use PCG with the initial iterate to be the previous time step (or sub-step). In table 3, we display the [min, median, max] iteration count for all 3000 PCG calls required to time step 10 seconds. As in [5], we expect the number of iterations to not increase as we refine the mesh or increase $p$. Indeed, we see that the median iteration counts in table 3 is *the same* as

[5] Timings were done using Python 3 with the key kernels from [2] written in Cython on an Ryzen 5 1600 processor and 16GB of Ram

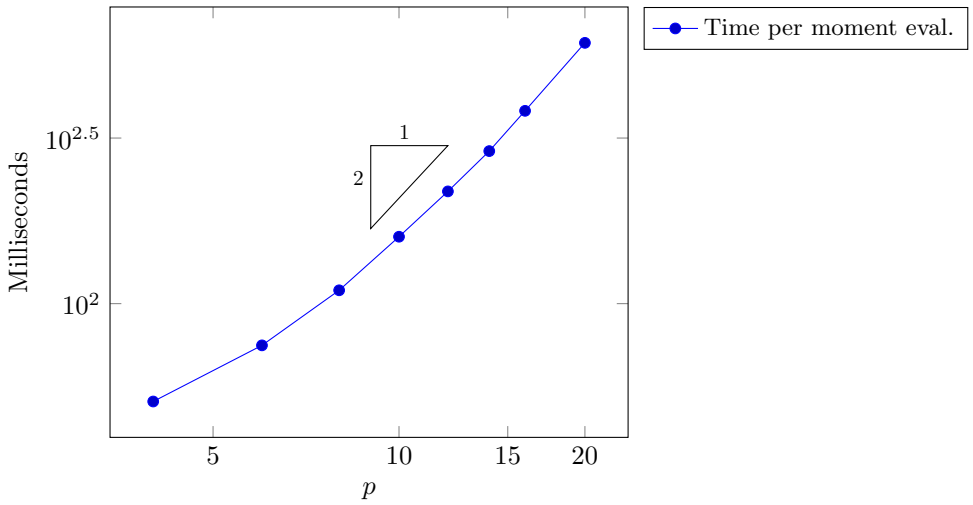
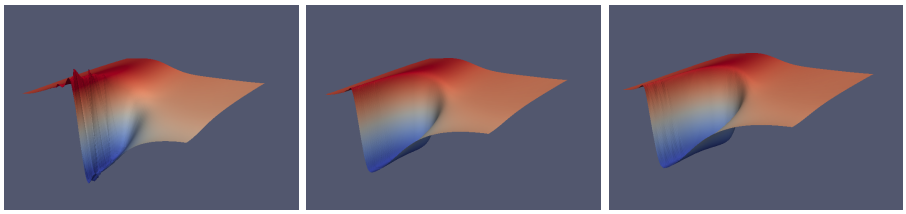



Figure 12: The average time required to compute the non-linearity in the Brusselator system is plotted on a log-log axis. We note that for the orders we are examining, the growth is $\mathcal{O}(p^2)$ rather than the asymptotic growth of $\mathcal{O}(p^3)$ from [2]. The asymptotic growth is observed for $p > 30$ (see [2]).

the iteration counts as in the linear wave equation considered in [5]; this is not an unexpected result as only the residuals have changed from the linear heat equation.

While the above result is certainly favorable, the case of explicit time-stepping allows for the use of the preconditioner of just the Schur complement as described in section 4.7. In table 4, we display the iteration count of solving the Schur complement (i.e. the iteration counts of line 5 of algorithm 11) in the period of 10 seconds for solving the sine-Gordon equation. We note that the iteration count does not increase as we refine $h$ or $p$ which we prove in section 7.1.

Finally, we will use the Sine-Gordon example to demonstrate that PCG is achieving the required accuracy. In section 5.2, we plot the residual of each iteration from PCG of the first linear solve at $t = 0$ for 64 elements; the residual decreases quite nicely and we achieve a tolerance of $10^{-9}$ easily. In fact, we note that the number of iterations decreases as $p$ increases which matches table 3



(a) $p = 4$ (b) $p = 8$ (c) $p = 16$

Figure 13: Plots of the solution to the Brusselator example at $t = 5$ with the $z$-axis scaled by a factor of .1 and time-steps as in table 2.

and table 4.

Table 3: Table illustrates the performance of the preconditioned iterative method of the mass matrix at each time step by displaying the [min, median, max] iteration count of all 3000 PCG solves from using the Nyström method for a period of 10 seconds with a $\Delta t = .01$ in a uniformly triangulated square for the sine-Gordon equation.

| Order | 16 Elements | 64 Elements | 256 Elements |
|---|---|---|---|
| 4 | [21, 26, 32] | [20, 25, 34] | [17, 23, 31] |
| 8 | [17, 23, 29] | [16, 21, 30] | [16, 21, 26] |
| 12 | [17, 22, 27] | [16, 18, 26] | [17, 17, 24] |
| 16 | [16, 18, 25] | [15, 18, 24] | [15, 15, 22] |
| 20 | [16, 18, 24] | [15, 15, 23] | |

### *5.3. Boundary Layer Problems*

We now illustrate the use of the preconditioner in the numerical solution of a boundary layer problem. Let $u(x, y)$ be the solution to the problem as described in section 2.3 with $f = 1$. In [5], we remarked that our mass preconditioner allows for needle elements, hence we showcase this capability by using the mesh as shown in fig. 4.

In fig. 15, we plot the condition number of the preconditioned system $\mathbf{P}^{-1/2}(\mathbf{M}+\varepsilon^2\mathbf{S})\mathbf{P}^{-1/2}$. We observe that the growth of the condition numbers grows as $p^2$ as eq. (9) suggests, and that for $\varepsilon$ small enough, that the condition numbers do

Table 4: Table illustrates the performance of the preconditioner *based on the Schur complement* of the mass matrix at each time step by displaying the [min, median, max] iteration count of the Schur complement solve (line 5 of algorithm 11) from using the Nyström method for a period of 10 seconds with a $\Delta t = .01$ in a uniformly triangulated square for the sine-Gordon equation.

| Order | 16 Elements | 64 Elements | 256 Elements |
|---|---|---|---|
| 4 | [22, 27, 33] | [21, 26, 35] | [18, 24, 32] |
| 8 | [18, 24, 30] | [17, 22, 31] | [17, 22, 27] |
| 12 | [18, 23, 28] | [17, 19, 27] | [17, 18, 25] |
| 16 | [17, 19, 26] | [1, 19, 25] | [16 ,16, 23] |
| 20 | [1, 18, 25] | [1, 16, 24] | |

not depend on $\varepsilon$.

## 6. Conclusion

The current work described the efficient implementation of a $p$-version mass matrix preconditioner using the Bernstein basis, alongside useful post-processing procedures such as visualization and gradient evaluations. Of particular note is an algorithm to invert the interior blocks of the mass matrix in $\mathcal{O}(p^3)$ operations. This allowed us to perform the preconditioning step with a total cost of $\mathcal{O}(p^3)$, hence, combined with the results from [2], allows for one to construct the mass and stiffness matrices, time step, and perform post-processing of nonlinear transient problems all with a cost of $\mathcal{O}(p^3)$. While preconditioning the mass matrix will offer no advantages for problems where only the stiffness matrix is present, we also showed that certain elliptic problems such as the singularly perturbed problem can be handled by a preconditioner for the mass matrix. Some of the algorithms does extend naturally to tetrahedrons such as the de Casteljau algorithm, and the Bernstein basis matrix construction and multiplies from [2]. Unfortunately, neither the mass preconditioner in 3D or the interior inversion algorithm extend as easily to 3D and will be the subject of a forthcoming work.

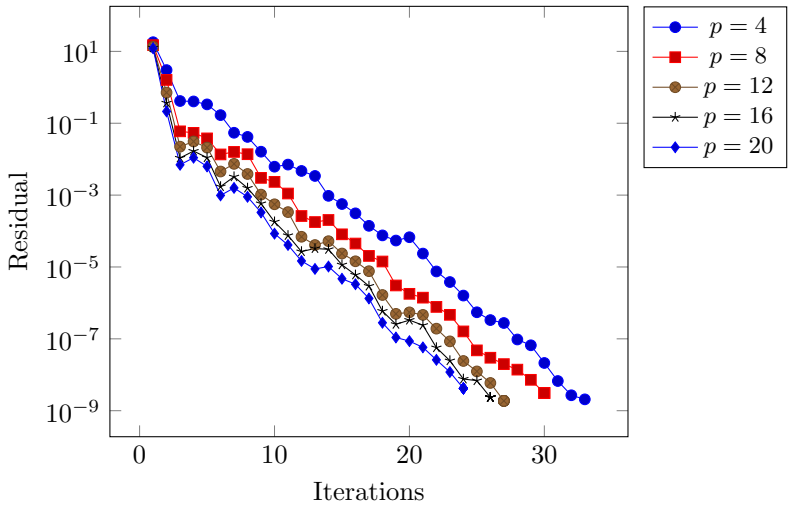



Figure 14: Plot of the residuals resulting from the preconditioned conjugate gradient method applied to the Sine-Gordon example at $t = 0$ and 64 elements.

## 7. Appendix

### *7.1. Schur Complement Preconditioner*

In this section, we present a short proof that the preconditioner for the Schur complement (algorithm 10) has bounded condition numbers. Like in [5], it suffices to show the result on the reference triangle. Let $\widetilde{X}_B, \widetilde{X}_V$ and $\widetilde{X}_{E_i}, i = 1, 2, 3$ be the minimal $L^2$ extension space as defined in §5 of [5], with the inner-products as $a_V(\cdot, \cdot)$ and $a_{E_i}(\cdot, \cdot)$ from the same section.

The additive Schwarz method preconditioner which arises is given $\widetilde{f} \in \widetilde{X}_B$, find $u$ as follows:

1. $u_V \in \widetilde{X}_V : a_V(u_V, v_V) = (\widetilde{f}, v_V) \quad \forall v_V \in \widetilde{X}_V$.
2. For $i = 1, 2, 3$, $u_{E_i} \in \widetilde{X}_{E_i} : a_{E_i}(u_{E_i}, v_{E_i}) = (\widetilde{f}, v_{E_i}) \quad \forall v_{E_i} \in \widetilde{X}_{E_i}$.
3. $u := u_V + \sum_{i=1}^{3} u_{E_i}$ is our solution.

Note that it is simply what we had in [5], except the interior solve is not there; this leads to a simple corollary.

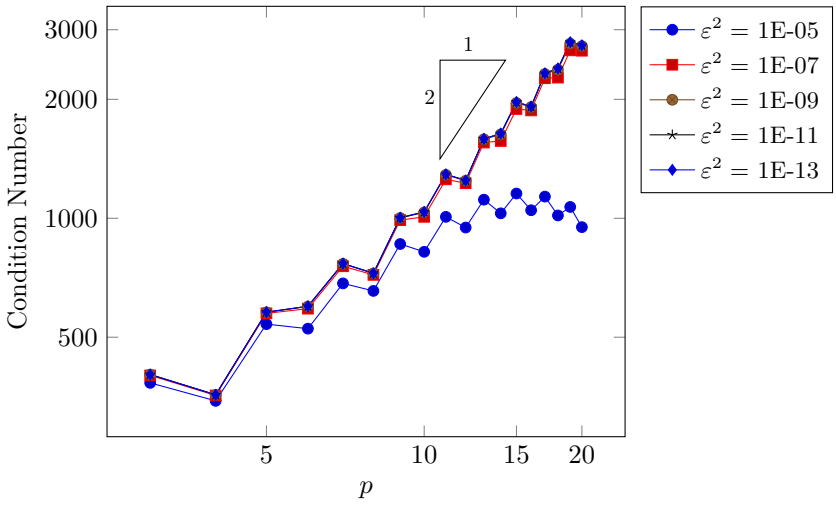



Figure 15: The condition numbers of the preconditioned system for the boundary layer problem using the mesh in fig. 4 are displayed for varying $\varepsilon$ and $p$ in a log-log scale. We note that the growth of the condition number satisfies the Schmidt's inequality estimate, and that as we decrease $\varepsilon$, the condition number seems to converge to a curve.

**Corollary 7.0.1.** *The abstract additive Schwarz method defined above corresponds to algorithm 10 under the Bernstein basis. Furthermore, there exists a constant $C$ independent of $h, p$ such that cond$(\ddot{\mathbf{P}}^{-1}\ddot{\mathbf{S}}) \leq C$.*

*Proof.* Let us first prove that the abstract ASM has uniform condition number. We see that Lemma 5.3, Lemma 5.4 and Theorem 5.5 can be easily modified to reflect the ASM method above by removing the interior portions from each of the statements; hence this is simply a consequence of Theorem 2.7 of [47].

Finally, applying the exact same techniques from section 4.4 and section 4.5, keeping in mind that we are given $\widetilde{f} \in \widetilde{X}_B$, we see that the ASM method corresponds to algorithm 10.

□

### *7.2. Dirichlet boundary condition*

The enforcement of Dirichlet boundary conditions is trivial to implement for the preconditioner. In algorithm 9, before the interior correction term (line

6), simply set the degrees of freedom in $\vec{x}_V, \vec{x}_E$ corresponding to the Dirichlet boundary condition equal to the appropriate Bernstein basis values; in our case for the boundary layer case study, this was simply 0.

The more mathematically accurate way would be to modify the diagonal scaling matrices $\mathbf{D}_{VV}, \mathbf{D}_{EE}$ to be 1 at the Dirichlet boundary condition dofs and also use the modified mass matrix (zeroing out the rows and columns and leaving a one on the diagonal) for Dirichlet boundary conditions, the fact that the edge solve and vertex solve are diagonal allows us to use the procedure above.

### *7.3. Degree Raising Algorithms*

The degree raising formula for the 1D Bernstein polynomials is easily derived:

$$B_i^p(x) = (\lambda_1 + \lambda_2)B_i^p(x) = \frac{p+1-i}{p+1}B_i^{p+1}(x) + \frac{i+1}{p+1}B_{i+1}^{p+1}(x). \tag{25}$$

This allows us to express a Bernstein basis polynomial of degree $p$ as one of degree $p+1$ as such

$$\sum_{i=0}^{p} c_i^p B_i^p(x) = \sum_{i=0}^{p+1} c_i^{p+1} B_i^{p+1}(x).$$

The following subroutine computes $c_i^{p+1}$ in $\mathcal{O}(p)$:

**Algorithm 12** Degree Raising Operator

**Require:** $\vec{c}$ corresponding to the B-net of the polynomial of degree $p$

1: **function** DEGREERAISE1D($\vec{c}$)
2: $\vec{o} = \text{zeros}(p+2)$ ▷ Coefficients for degree $p+1$
3: **for** $i = 0, \ldots, p$ **do**
4: $o[i] += (p+1-i)c[i]/(p+1)$
5: $o[i+1] += (i+1)c[i]/(p+1)$
6: **end for**
7: **return** $\vec{o}$
8: **end function**

An equally useful operation is the "degree-lowering operation," which is the opposite of the degree raising operation. This is only used when we are working with inner-products; for example, for a function $g$, we can deduce $(g, B_i^3)$ for $i = 0, \ldots, 3$ given $(g, B_j^4)$ for $j = 0, \ldots, 4$. The degree lowering operator also has a cost of $\mathcal{O}(p)$ as it is simply the degree raising operator backwards.

**Algorithm 13** Degree Lowering Operator

**Require:** $\vec{c}$ corresponding to the inner-products $(f, B_i^{p+1})$ of Bernstein polynomial of degree $p+1$

```
function DegreeLower(c⃗)
    o⃗ = zeros(p + 1)                          ▷ Coefficients for degree p
    for i = 0, ..., p do
        o[i] = ((p + 1 − i)c[i] + (i + 1)c[i + 1])/(p + 1)
    end for
    return o⃗
end function
```

In two dimensions, the Bernstein polynomials also satisfy a degree raising operation. Let $e_k \in \mathbb{R}^3$ be one at the $k$th index, and zero elsewhere. We have that

$$B_\alpha^p = (\lambda_1 + \lambda_2 + \lambda_3) B_\alpha^p = \frac{\alpha_1 + 1}{p+1} B_{\alpha+e_1}^{p+1} + \frac{\alpha_2 + 1}{p+1} B_{\alpha+e_2}^{p+1} + \frac{\alpha_3 + 1}{p+1} B_{\alpha+e_3}^{p+1}.$$

If we store the control points $\{c_\alpha^p\}$ in a 2D array, then the following algorithm performs degree raising in $\mathcal{O}(p^2)$:

**Algorithm 14** 2D Degree Raising Operator

**Require:** $\mathbf{c}$ array of the B-net of the polynomial of degree $p$

**function** DegreeRaise2D($\mathbf{c}$)
  $\mathbf{o} = \text{zeros}((p+2, p+2))$ ▷ Coefficients for degree $p+1$
  **for** $i = 0, \ldots, p$ **do**
    **for** $j = 0, \ldots, p-i$ **do**
      $k = p - i - j$
      $\mathbf{o}[i, j] += (k+1)/(p+1) * \mathbf{c}[i, j]$
      $\mathbf{o}[i+1, j] += (i+1)/(p+1) * \mathbf{c}[i, j]$
      $\mathbf{o}[i, j+1] += (j+1)/(p+1) * \mathbf{c}[i, j]$
    **end for**
  **end for**
  **return** $\vec{o}$
**end function**

There are many more mathematical and computational properties of Bernstein polynomials which we do not not need; a general reference can be found in [18].

*7.4. Proofs for section 4.3*

In this subsection, we prove the lemmas used in section 4.3. We first prove lemma 4.2.

*Proof of lemma 4.2.* We begin observing that eq. (12) gives

$$\begin{aligned} P_{m-r}^{2r+5,2}(t) &= \sum_{\alpha_3=0}^{m-r} (-1)^{m-r-\alpha_3} \frac{\binom{m+r+5}{\alpha_3}\binom{m-r+2}{m-r-\alpha_3}}{\binom{m-r}{\alpha_3}} B_{\alpha_3}^{m-r}(t) \\ &= \sum_{\alpha_3=0}^{m-r} \nu_{\alpha_3}^{mr} \binom{m}{\alpha_3} (\lambda_1 + \lambda_2)^{m-r-\alpha_3} \lambda_3^{\alpha_3}, \end{aligned}$$

and

$$P_r^{(2,2)}(s)\left(\frac{1-t}{2}\right)^r = \sum_{\alpha_2=0}^{r}(-1)^{r-\alpha_2}\frac{\binom{r+2}{\alpha_2}\binom{r+2}{r-\alpha_2}}{\binom{r}{\alpha_2}}B_{\alpha_2}^r(s)\left(\frac{1-t}{2}\right)^r$$
$$= \sum_{\alpha_2=0}^{r}\gamma_{\alpha_2}^r\binom{r}{\alpha_2}\lambda_1^{r-\alpha_2}\lambda_2^{\alpha_2}.$$

Using the binomial formula and with the convention $\gamma_i^r = 0$ for $i < 0$ and $i > r$, we can write

$$(\lambda_1+\lambda_2)^{m-r-\alpha_3}P_r^{(2,2)}(s)\left(\frac{1-t}{2}\right)^r$$
$$= \sum_{l=0}^{m-r-\alpha_3}\binom{m-r-\alpha_3}{l}\lambda_1^{m-r-\alpha_3-l}\lambda_2^l\sum_{\alpha_2=0}^{r}\gamma_{\alpha_2}^r\binom{r}{\alpha_2}\lambda_1^{r-\alpha_2}\lambda_2^{\alpha_2}$$
$$= \sum_{l=0}^{m-r-\alpha_3}\binom{m-r-\alpha_3}{l}\sum_{\alpha_2=l}^{r+l}\gamma_{\alpha_2-l}^r\binom{r}{\alpha_2-l}\lambda_1^{m-\alpha_3-\alpha_2}\lambda_2^{\alpha_2}$$
$$= \sum_{\alpha_2=0}^{m-\alpha_3}\left(\sum_{l=0}^{m-r-\alpha_3}\gamma_{\alpha_2-l}^r\frac{\binom{m-r-\alpha_3}{l}\binom{r}{\alpha_2-l}}{\binom{m-\alpha_3}{\alpha_2}}\right)\binom{m-\alpha_3}{\alpha_2}\lambda_1^{m-\alpha_3-\alpha_2}\lambda_2^{\alpha_2}.$$

Therefore,

$$P_r^{(2,2)}(s)\left(\frac{1-t}{2}\right)^r P_{m-r}^{2r+5,2}(t)$$
$$= \sum_{\alpha_3=0}^{m-r}\nu_{\alpha_3}^{mr}\sum_{\alpha_2=0}^{m-\alpha_3}\gamma_{\alpha_2}^{r,m-\alpha_3}\binom{m-\alpha_3}{\alpha_2}\binom{m}{\alpha_3}\lambda_1^{m-\alpha_3-\alpha_2}\lambda_2^{\alpha_2}\lambda_3^{\alpha_3}$$
$$= \sum_{\alpha_3=0}^{m-r}\nu_{\alpha_3}^{mr}\sum_{\alpha_2=0}^{m-\alpha_3}\gamma_{\alpha_2}^{r,m-\alpha_3}B_\alpha^m(x,y),$$

which proves the identity. □

*7.4.1. Proofs for the recursive computation of $S^{mr}$ (and $Q_{\alpha_3}^{mr}$)*

We first show an auxiliary result concerning the coefficients $\gamma_{\alpha_2}^{rj}$ from lemma 4.2.

**Lemma 7.1.** *Consider $\gamma_{\alpha_2}^{rj}$, $j = r,\dots,m$ and $r = 0,\dots,m$ introduced in lemma 4.2. Then,*

$$\gamma_{\alpha_2}^{r,j+1} = \frac{\alpha_2}{j+1}\gamma_{\alpha_2-1}^{r,j} + \frac{j+1-\alpha_2}{j+1}\gamma_{\alpha_2}^{r,j}, \quad \text{for } \alpha_2 = 0,\dots,j+1,$$

*i.e. $\vec{\gamma}^{r,j+1} = \mathcal{R}(\vec{\gamma}^{r,j})$ for $j = r,\dots,m-1$., where $\mathcal{R}$ denotes the degree raising operator in one dimension (algorithm 12).*

*Proof.* By the properties of the binomial coefficients, we have that

$$\begin{aligned}\frac{\alpha_2}{j+1}\gamma^{r,j}_{\alpha_2-1}+\frac{j+1-\alpha_2}{j+1}\gamma^{r,j}_{\alpha_2} &= \sum_{l=0}^{j-r}\gamma^{r}_{\alpha_2-1-l}\frac{\binom{j-r}{l-1}\binom{r}{\alpha_2-1-l}}{\binom{j+1}{\alpha_2}}+\sum_{l=0}^{j-r}\gamma^{r}_{\alpha_2-l}\frac{\binom{j-r}{l}\binom{r}{\alpha_2-l}}{\binom{j+1}{\alpha_2}}\\ &= \sum_{l=1}^{j+1-r}\gamma^{r}_{\alpha_2-l}\frac{\binom{j-r}{l-1}\binom{r}{\alpha_2-l}}{\binom{j+1}{\alpha_2}}+\sum_{l=0}^{j-r}\gamma^{r}_{\alpha_2-l}\frac{\binom{j-r}{l}\binom{r}{\alpha_2-l}}{\binom{j+1}{\alpha_2}}\\ &= \sum_{l=0}^{j+1-r}\gamma^{r}_{\alpha_2-l}\frac{\binom{r}{\alpha_2-l}}{\binom{j+1}{\alpha_2}}\left(\binom{j-r}{l-1}+\binom{j-r}{l}\right)=\gamma^{r,j+1}_{\alpha_2}\end{aligned}$$

which completes the proof. □

**Lemma 7.2.**

$$Q^{mr}_{\alpha_3}=\frac{\alpha_3+1}{m+1}Q^{m+1r}_{\alpha_3+1}+\frac{m+1-\alpha_3}{m+1}Q^{m+1r}_{\alpha_3},$$

*for* $\alpha_3=0,\ldots,m$ *and* $r=0,\ldots,m$.

*Proof.* We use the two dimensional degree raise operator on the coefficients $\tilde{f}^m_\alpha$ and and lemma 7.1

$$\begin{aligned}Q^{mr}_{\alpha_3} &= \sum_{\alpha_1+\alpha_2=m-\alpha_3}\gamma^{r,m-\alpha_3}_{\alpha_2}\tilde{f}^m_\alpha\\ &= \sum_{\alpha_1+\alpha_2=m-\alpha_3}\gamma^{r,m-\alpha_3}_{\alpha_2}\left(\frac{\alpha_1+1}{m+1}\tilde{f}^{m+1}_{\alpha+e_1}+\frac{\alpha_2+1}{m+1}\tilde{f}^{m+1}_{\alpha+e_2}+\frac{\alpha_3+1}{m+1}\tilde{f}^{m+1}_{\alpha+e_3}\right)\\ &= \frac{m+1-\alpha_3}{m+1}\sum_{\alpha_1+\alpha_2=m+1-\alpha_3}\left(\frac{\alpha_1}{m+1-\alpha_3}\gamma^{r,m-\alpha_3}_{\alpha_2}+\frac{\alpha_2}{m+1-\alpha_3}\gamma^{r,m-\alpha_3}_{\alpha_2-1}\right)\tilde{f}^m_\alpha\\ &\quad+\frac{\alpha_3+1}{m+1}Q^{m+1\,r}_{\alpha_3+1}\\ &= \frac{m+1-\alpha_3}{m+1}\sum_{\alpha_1+\alpha_2=m+1-\alpha_3}\gamma^{r,m+1-\alpha_3}_{\alpha_2}\tilde{f}^{m+1}_\alpha+\frac{\alpha_3+1}{m+1}Q^{m+1\,r}_{\alpha_3+1}\\ &= \frac{m+1-\alpha_3}{m+1}Q^{m+1\,r}_{\alpha_3}+\frac{\alpha_3+1}{m+1}Q^{m+1\,r}_{\alpha_3+1}.\end{aligned}$$

□

*7.4.2. Proofs for the recursive computation of* $T^m_\beta$

We first need to prove a fact for the coefficients $a^{mr}_\alpha$.

**Lemma 7.3.** *Consider the coefficients $a_\alpha^{mr}$, $r = 0, \dots, m$. Then, the following identity holds*

$$(\alpha_1 + 3)a^{mr}_{\alpha+e_1} + (\alpha_2 + 3)a^{mr}_{\alpha+e_2} + (\alpha_3 + 3)a^{mr}_{\alpha+e_3} = 0,$$

*for $|\alpha| = m - 1$.*

*Proof.* We observe that the statement is indeed equivalent to

$$\begin{aligned}(m - \alpha_3 - \alpha_2 + 2)\gamma^{r,m-\alpha_3}_{\alpha_2} + (\alpha_2 + 3)\gamma^{r,m-\alpha_3}_{\alpha_2+1} &= -(\alpha_3 + 3)\frac{\nu^{mr}_{\alpha_3+1}}{\nu^{mr}_{\alpha_3}}\gamma^{r,m-\alpha_3-1}_{\alpha_2} \\ &= \frac{(m - \alpha_3 + r + 5)(m - \alpha_3 - r)}{m - \alpha_3}\gamma^{r,m-\alpha_3-1}_{\alpha_2},\end{aligned}$$

which we now prove.

For any $0 \le \alpha_2 \le m - 1$, we proceed by induction on $\alpha_3 = 0, \dots, m - 1 - r$ (equivalently $m - \alpha_3 = r + 1, \dots, m$) as $a_\alpha^{mr} = 0$ for $\alpha_3 > m - 1 - r$. We first prove the statement for $m - \alpha_3 = r + 1$, i.e., we prove the identity

$$(r + 3 - \alpha_2)\gamma^{r,r+1}_{\alpha_2} + (\alpha_2 + 3)\gamma^{r,r+1}_{\alpha_2+1} = \frac{2(r + 3)}{r + 1}\gamma^{r,r}_{\alpha_2}.$$

We first note that $\gamma^{r,r}_{\alpha_2} = \gamma^r_{\alpha_2}$. We note that

$$\gamma^r_{\alpha_2-1} = -\frac{(\alpha_2 + 2)}{(r + 3 - \alpha_2)}\gamma^r_{\alpha_2}, \quad \text{for } \alpha_2 = 1, \dots, r + 1,$$

and by lemma 7.1

$$\gamma^{r,r+1}_{\alpha_2} = \frac{\alpha_2}{r + 1}\gamma^r_{\alpha_2-1} + \frac{r + 1 - \alpha_2}{r + 1}\gamma^r_{\alpha_2}, \quad \text{for } \alpha_2 = 0, \dots, r + 1.$$

Thus, it follows

$$\begin{aligned}&(r + 3 - \alpha_2)\gamma^{r,r+1}_{\alpha_2} + (\alpha_2 + 3)\gamma^{r,r+1}_{\alpha_2+1} \\ &\quad = (r + 3 - \alpha_2)\left(\frac{\alpha_2}{r + 1}\gamma^r_{\alpha_2-1} + \frac{r + 1 - \alpha_2}{r + 1}\gamma^r_{\alpha_2}\right) + (\alpha_2 + 3)\left(\frac{\alpha_2 + 1}{r + 1}\gamma^r_{\alpha_2} + \frac{r - \alpha_2}{r + 1}\gamma^r_{\alpha_2+1}\right) \\ &\quad = \frac{2(r + 3)}{r + 1}\gamma^r_{\alpha_2}.\end{aligned}$$

We now assume the statement is true for $n > r$, i.e.,

$$(n + 2 - \alpha_2)\gamma^{r,n}_{\alpha_2} + (\alpha_2 + 3)\gamma^{r,n}_{\alpha_2+1} = \frac{(n + r + 5)(n - r)}{n}\gamma^{r,n-1}_{\alpha_2},$$

and we prove it for $n+1$, i.e., we prove the identity

$$(n+3-\alpha_2)\gamma_{\alpha_2}^{r,n+1}+(\alpha_2+3)\gamma_{\alpha_2+1}^{r,n+1}=\frac{(n+r+6)(n+1-r)}{n+1}\gamma_{\alpha_2}^{r,n}.$$

Arrangement of the inductive hypothesis gives

$$\gamma_{\alpha_2-1}^{r,n}=\frac{1}{(n+3-\alpha_2)}\left(\frac{(n+r+5)(n-r)}{n}\gamma_{\alpha_2-1}^{r,n-1}-(\alpha_2+2)\gamma_{\alpha_2}^{r,n}\right),$$
$$\gamma_{\alpha_2}^{r,n}=\frac{1}{(\alpha_2+3)}\left(\frac{(n+r+5)(n-r)}{n}\gamma_{\alpha_2}^{r,n-1}-(n+2-\alpha_2)\gamma_{\alpha_2}^{r,n}\right).$$

Thus using lemma 7.1 repeatedly, we have

$$\begin{aligned}
&(n+3-\alpha_2)\gamma_{\alpha_2}^{r,n+1}+(\alpha_2+3)\gamma_{\alpha_2+1}^{r,n+1}\\
=&\frac{1}{n+1}\left((n+3-\alpha_2)\left(\alpha_2\gamma_{\alpha_2-1}^{r,n}+(n+1-\alpha_2)\gamma_{\alpha_2}^{r,n}\right)+(\alpha_2+3)\left((\alpha_2+1)\gamma_{\alpha_2}^{r,n}+(n-\alpha_2)\gamma_{\alpha_2+1}^{r,n}\right)\right)\\
=&\frac{1}{n+1}\left(\alpha_2\Big(\frac{(n+r+5)(n-r)}{n}\gamma_{\alpha_2-1}^{r,n-1}-(\alpha_2+2)\gamma_{\alpha_2}^{r,n}\Big)+(n+3-\alpha_2)(n+1-\alpha_2)\gamma_{\alpha_2}^{r,n}\right.\\
&\left.+(\alpha_2+3)(\alpha_2+1)\gamma_{\alpha_2}^{r,n}+(n-\alpha_2)\Big(\frac{(n+r+5)(n-r)}{n}\gamma_{\alpha_2}^{r,n-1}-(n+2-\alpha_2)\gamma_{\alpha_2}^{r,n}\Big)\right)\\
=&\frac{1}{n+1}\left((n+r+5)(n-r)\Big(\frac{\alpha_2}{n}\gamma_{\alpha_2-1}^{r,n-1}+\frac{n-\alpha_2}{n}\gamma_{\alpha_2}^{r,n-1}\Big)+\gamma_{\alpha_2}^{r,n}(2n+6)\right)\\
=&\frac{1}{n+1}\left((n+r+5)(n-r)\gamma_{\alpha_2}^{r,n}+\gamma_{\alpha_2}^{r,n}(2n+6)\right)\\
=&\frac{(n+r+6)(n+1-r)}{n+1}\gamma_{\alpha_2}^{r,n},
\end{aligned}$$

which completes the proof. □

**Lemma 7.4.** *The following identity hold for* $|\beta|=m-1$

$$T_{\beta+e_3}^m=-\left(\frac{\beta_1+3}{\beta_3+3}T_{\beta+e_1}^m+\frac{\beta_2+3}{\beta_3+3}T_{\beta+e_2}^m\right).$$

*Proof.* By definition of $T_\beta^m$ for $|\beta|=m-1$ and applying lemma 7.3, it follows

$$\begin{aligned}
T_{\beta+e_3}^m&=\sum_{r=0}^{m}S^{mr}a_{\beta+e_3}^{mr}\\
&=-\sum_{r=0}^{m}S^{mr}\left(\frac{\beta_1+3}{\beta_3+3}a_{\beta+e_1}^{mr}+\frac{\beta_2+3}{\beta_3+3}a_{\beta+e_2}^{mr}\right)\\
&=-\left(\frac{\beta_1+3}{\beta_3+3}\sum_{r=0}^{m}S^{mr}a_{\beta+e_1}^{mr}+\frac{\beta_2+3}{\beta_3+3}\sum_{r=0}^{m}S^{mr}a_{\beta+e_2}^{mr}\right).
\end{aligned}$$

□

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55 of *National Bureau of Standards Applied Mathematics Series*. For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 1964.

[2] Mark Ainsworth, Gaelle Andriamaro, and Oleg Davydov. Bernstein-Bézier finite elements of arbitrary order and optimal assembly procedures. *SIAM J. Sci. Comput.*, 33(6):3087–3109, 2011.

[3] Mark Ainsworth, Gaelle Andriamaro, and Oleg Davydov. A Bernstein-Bézier basis for arbitrary order Raviart-Thomas finite elements. *Constr. Approx.*, 41(1):1–22, 2015.

[4] Mark Ainsworth and Joe Coyle. Conditioning of hierarchic $p$-version Nédélec elements on meshes of curvilinear quadrilaterals and hexahedra. *SIAM J. Numer. Anal.*, 41(2):731–750, 2003.

[5] Mark Ainsworth and Shuai Jiang. Preconditioning the Mass Matrix for High Order Finite Element Approximation on Triangles. *SIAM J. Numer. Anal.*, 57(1):355–377, 2019.

[6] Thomas Apel. *Anisotropic finite elements: local estimates and applications*. Advances in Numerical Mathematics. B. G. Teubner, Stuttgart, 1999.

[7] Ivo Babuška and Anthony Miller. The post-processing approach in the finite element method— Part 1: calculation of displacements, stresses and other higher derivatives of the displacements. *International Journal for numerical methods in engineering*, 20(6):1085–1109, 1984.

[8] I. Babuška and B. Q. Guo. Regularity of the solution of elliptic problems with piecewise analytic data. I. Boundary value problems for linear elliptic equation of second order. *SIAM J. Math. Anal.*, 19(1):172–203, 1988.

[9] A Barone, F Esposito, CJ Magee, and AC Scott. Theory and applications of the sine-Gordon equation. *La Rivista del Nuovo Cimento (1971-1977)*, 1(2):227–267, 1971.

[10] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring. I. *Math. Comp.*, 47(175):103–134, 1986.

[11] A. G. Bratsos. The solution of the two-dimensional sine-Gordon equation using the method of lines. *J. Comput. Appl. Math.*, 206(1):251–277, 2007.

[12] Elaine Cohen and Larry L. Schumaker. Rates of convergence of control polygons. *Comput. Aided Geom. Design*, 2(1-3):229–235, 1985. Surfaces in CAGD '84 (Oberwolfach, 1984).

[13] Wolfgang Dahmen. Subdivision algorithms converge quadratically. *J. Comput. Appl. Math.*, 16(2):145–158, 1986.

[14] Leszek Demkowicz. *Computing with hp-adaptive finite elements. Vol. 1.* Chapman & Hall/CRC Applied Mathematics and Nonlinear Science Series. Chapman & Hall/CRC, Boca Raton, FL, 2007. One and two dimensional elliptic and Maxwell problems, With 1 CD-ROM (UNIX).

[15] Z. Ditzian. Multivariate Bernstein and Markov inequalities. *J. Approx. Theory*, 70(3):273–283, 1992.

[16] P. G. Drazin and R. S. Johnson. *Solitons: an introduction.* Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 1989.

[17] Moshe Dubiner. Spectral methods on triangles and other domains. *J. Sci. Comput.*, 6(4):345–390, 1991.

[18] Gerald E. Farin. *Curves and Surfaces for CAGD: A Practical Guide.*, volume 5th ed of *The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, 2002.

[19] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Comput. Aided Geom. Design*, 5(1):1–26, 1988.

[20] Rida T. Farouki. The Bernstein polynomial basis: a centennial retrospective. *Comput. Aided Geom. Design*, 29(6):379–419, 2012.

[21] Jean Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. Morgan Kaufmann, 2000.

[22] Ron Goldman. *Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling*. Elsevier, 2002.

[23] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[24] A. E. Green and W. Zerna. *Theoretical elasticity*. Dover Publications, Inc., New York, second edition, 1992.

[25] Ernst Hairer. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 2011.

[26] Yannis Haralambous. Parametrization of postscript fonts through metafont: an alternative to adobe multiple master fonts. *Electronic Publishing*, 6(3):145–157, 1993.

[27] Shi-Min Hu. Conversion between triangular and rectangular Bézier patches. *Comput. Aided Geom. Design*, 18(7):667–671, 2001. Special issue Pierre Bézier.

[28] George Em Karniadakis and Spencer J. Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, second edition, 2005.

[29] Boris N. Khoromskij and Gabriel Wittum. *Numerical solution of elliptic differential equations by reduction to the interface*, volume 36 of *Lecture*

*Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2004.

[30] Robert C. Kirby. Fast inversion of the simplicial Bernstein mass matrix. *Numer. Math.*, 135(1):73–95, 2017.

[31] Tom Lyche and Karl Scherer. On the $p$-norm condition number of the multivariate triangular Bernstein basis. *J. Comput. Appl. Math.*, 119(1-2):259–273, 2000. Dedicated to Professor Larry L. Schumaker on the occasion of his 60th birthday.

[32] Jean-François Maitre and Olivier Pourquier. Conditionnements et préconditionnements diagonaux pour la $p$-version des méthodes d'éléments finis pour des problèmes elliptiques du second ordre. *C. R. Acad. Sci. Paris Sér. I Math.*, 318(6):583–586, 1994.

[33] Jan Mandel and G. Scott Lett. Domain decomposition preconditioning for $p$-version finite elements with high aspect ratios. *Appl. Numer. Math.*, 8(4-5):411–425, 1991.

[34] Jens M. Melenk. *hp-finite element methods for singular perturbations*, volume 1796 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2002.

[35] Elwood T. Olsen and Jim Douglas, Jr. Bounds on spectral condition numbers of matrices arising in the $p$-version of the finite element method. *Numer. Math.*, 69(3):333–352, 1995.

[36] Steven A. Orszag. Spectral methods for problems in complex geometries. *J. Comput. Phys.*, 37(1):70–92, 1980.

[37] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Mathematics and Visualization. Springer-Verlag, Berlin, 2002.

[38] Jean-François Remacle, Nicolas Chevaugeon, Émilie Marchandise, and Christophe Geuzaine. Efficient visualization of high-order finite elements. *Internat. J. Numer. Methods Engrg.*, 69(4):750–771, 2007.

[39] MP Rossow and IN Katz. Hierarchal finite elements and precomputed arrays. *International Journal for Numerical Methods in Engineering*, 12(6):977–999, 1978.

[40] Steven J. Ruuth. Implicit-explicit methods for reaction-diffusion problems in pattern formation. *J. Math. Biol.*, 34(2):148–176, 1995.

[41] Ch. Schwab. *p- and hp-finite element methods*. Numerical Mathematics and Scientific Computation. The Clarendon Press, Oxford University Press, New York, 1998. Theory and applications in solid and fluid mechanics.

[42] Christoph Schwab and Manil Suri. The $p$ and $hp$ versions of the finite element method for problems with boundary layers. *Math. Comp.*, 65(216):1403–1429, 1996.

[43] Spencer J. Sherwin and George Em. Karniadakis. A new triangular and tetrahedral basis for high-order ($hp$) finite element methods. *Internat. J. Numer. Methods Engrg.*, 38(22):3775–3802, 1995.

[44] Dave Shreiner. *OpenGL reference manual: The official reference document to OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[45] Barna Szabó and Ivo Babuška. *Finite element analysis*. A Wiley-Interscience Publication. John Wiley & Sons, Inc., New York, 1991.

[46] Andrea Toselli and Xavier Vasseur. A numerical study on Neumann-Neumann and FETI methods for $hp$ approximations on geometrically refined boundary layer meshes in two dimensions. *Comput. Methods Appl. Mech. Engrg.*, 192(41-42):4551–4579, 2003.

[47] Andrea Toselli and Olof Widlund. *Domain decomposition methods—algorithms and theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.

[48] Miki Wadati, Heiji Sanuki, and Kimiaki Konno. Relationships among inverse method, Bäcklund transformation and an infinite number of conservation laws. *Progr. Theoret. Phys.*, 53:419–436, 1975.

[49] Edward L Wilson. The static condensation algorithm. *International Journal for Numerical Methods in Engineering*, 8(1):198–203, 1974.

[50] Lanlan Yan, Xuli Han, and Jiongfeng Liang. Conversion between triangular Bézier patches and rectangular Bézier patches. *Applied Mathematics and Computation*, 232:469–478, 2014.