Projection methods for the incompressible Navier–Stokes equations using high-order hybridizable discontinuous Galerkin schemes

Corbin Foucart[a], C. Mirabito[a], P. J. Haley, Jr.[a], Pierre F.J. Lermusiaux[a,*]

[a]*Department of Mechanical Engineering, Computational Science and Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, 02139, MA, USA*

Abstract

abstract.tex

Keywords: Nonhydrostatic modeling, Finite element methods, Discontinuous Galerkin methods

1. Introduction

In [4], the authors investigate the stability and robustness of DG discretizations of several projection methods. They compare fully-implicit, high-order dual splitting [8], and pressure-correction schemes, but weirdly discretize the advection term implicitly for the pressure-correction, making the system nonlinear. So not exactly a fair comparison, since solving a nonlinear problem defeats the purpose of a projection method.

The numerical experiments in [4] suggest that the high-order, dual splitting scheme

1.1. Incompressible Navier–Stokes

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) - \nabla \cdot (\nu \nabla \boldsymbol{u}) + \nabla p &= \boldsymbol{f} \qquad \text{on } \Omega \times [0,T] \\ \nabla \cdot \boldsymbol{u} &= 0 \qquad \text{on } \Omega \times [0,T] \end{aligned} \tag{1}$$

Th incompressible Navier–stokes equations are subject to the initial condition

$$\boldsymbol{u}(\boldsymbol{x}, t=0) = \boldsymbol{u}_0 \text{ on } \Omega, \tag{2}$$

where $\boldsymbol{u}_0$ is divergence-free. We denote the outward-facing unit normal vector as $\boldsymbol{n}$. On the boundary Γ, we prescribe Dirichlet and Neumann conditions on Γ_D and Γ_N, respectively, such that $\Gamma_D \cup \Gamma_N = \Gamma$.

$$\boldsymbol{u} = \boldsymbol{g}_D^u \qquad \text{on } \Gamma_D \times [0,T] \tag{3}$$

$$(\boldsymbol{F}_v(\boldsymbol{u}) - p\boldsymbol{I}) \cdot \boldsymbol{n} = \boldsymbol{g}_N^{\text{stress}} \qquad \text{on } \Gamma_D \times [0,T] \tag{4}$$

Where $\boldsymbol{F}_v(\boldsymbol{u})$ is a representation of the viscous flux, usually given as $\boldsymbol{F}_v(\boldsymbol{u}) = \nu \nabla \boldsymbol{u}$. The operator splitting associated with projection methods will necessitate splitting the boundary condition as well, so we decompose the stress condition into viscous and pressure components $\boldsymbol{g}_N^{\text{stress}} = \boldsymbol{g}_N^u - g_N^p \boldsymbol{n}$ and prescribe them separately, following [4],

$$\boldsymbol{F}_v(\boldsymbol{u}) \cdot \boldsymbol{n} = \boldsymbol{g}_N^u \qquad \text{on } \Gamma_N, \tag{5}$$

$$p = g_N^p \qquad \text{on } \Gamma_N. \tag{6}$$

We use the Rothe method, handling the temporal discretization and operator splitting before spatial discretization. For time integration, we apply backward differentiation formula (BDF) for all schemes in this paper.

*Corresponding author,
Email addresses: foucartc@mit.edu (Corbin Foucart), pierrel@mit.edu (Pierre F.J. Lermusiaux)

2. Projection methods

An overview of the temporal discretization and operator splitting for pressure-correction schemes is given in [5]. Results from [5]: (1) under certain smoothness requirements on the solution, the nonlinear advection term in the Navier–Stokes equations does not affect the convergence rates of the splitting errors, and they treat it explicitly.

2.0.1. Velocity predictor step

An intermediate predictor velocity $\bar{\boldsymbol{u}}$ is calculated by solving the momentum equation with an explicit extrapolation of the pressure gradient term and explicit treatment of the advection term

$$\frac{\beta_s \bar{\boldsymbol{u}} - \sum_{i=0}^{s_u-1} \left(\beta_i \boldsymbol{u}^{n-i}\right)}{\Delta t} - \nabla \cdot (\nu \nabla \bar{\boldsymbol{u}}) = - \sum_{i=0}^{s_p-1} \left(\gamma_i \nabla p^{n-i}\right) - \nabla \cdot (\boldsymbol{u}^n \otimes \boldsymbol{u}^n) + \boldsymbol{f}(t_{n+1}), \tag{7}$$

where the boundary conditions for the predictor velocity are

$$\begin{aligned} \bar{\boldsymbol{u}} &= \boldsymbol{g}_D^u(t_{n+1}) \quad \text{on } \Gamma_D, \\ (\nu \nabla \bar{\boldsymbol{u}}) \cdot \boldsymbol{n} &= \boldsymbol{g}_N^u(t_{n+1}) \quad \text{on } \Gamma_N. \end{aligned} \tag{8}$$

2.0.2. Pressure corrector step

The second step involves computing a correction δp^{k+1} to the pressure by solving

$$-\nabla^2 \delta p^{k+1} = -\frac{\beta_s}{\Delta t} \nabla \cdot \bar{\boldsymbol{u}}, \tag{9}$$

subject to the boundary conditions

$$\nabla \delta p^{n+1} \cdot \boldsymbol{n} = 0 \quad \text{on } \Gamma_D, \tag{10}$$

$$\delta p^{n+1} = g_p(t_{n+1}) - \sum_{i=0}^{s_p-1} \left(\beta_i g_p(t_{n-i})\right) \quad \text{on } \Gamma_N. \tag{11}$$

The pressure Poisson equation (9) is obtained by writing the intermediate velocity $\bar{\boldsymbol{u}}$ in terms of a Helmholtz decomposition consisting of a solenoidal component $\boldsymbol{u}$ (since $\nabla \cdot \boldsymbol{u} = 0$) and irrotational component $\nabla \delta p^{n+1}$,

$$\frac{\beta_s}{\Delta t} \boldsymbol{u}^{n+1} + \nabla \delta p^{n+1} = \frac{\beta_s}{\Delta t} \bar{\boldsymbol{u}}, \tag{12}$$

$$\delta p^{n+1} = p^{n+1} - \sum_{i=0}^{s_p-1} \left(\beta_i p^{n-i}\right) + \chi \nu \nabla \cdot \bar{\boldsymbol{u}}, \tag{13}$$

then taking the divergence of equation (12), making use of the divergence-free constraint $\nabla \cdot \boldsymbol{u}^{n+1} = 0$. Taking $\chi = 0$ corresponds to the standard formulation and $\chi = 1$ the rotational formulation of the method, respectively [5]. We consider the rotational formulation hereafter.

2.0.3. Projection step

In the third step, the velocity $\boldsymbol{u}^{n+1}$ and pressure p^{n+1} at time t_{n+1} are obtained by performing updates amounting to the projection of $\bar{\boldsymbol{u}}$ onto the space of divergence-free vector fields

$$\boldsymbol{u}^{n+1} = \bar{\boldsymbol{u}} - \frac{\Delta t}{\beta_s} \nabla \delta p^{n+1}, \tag{14}$$

$$p^{n+1} = \sum_{i=0}^{s_p-1} \left(\beta_i p^{n-i}\right) + \delta p^{n+1} - \nu \nabla \cdot \bar{\boldsymbol{u}}, \tag{15}$$

Theoretical rates of convergence for the pressure-corrector schemes are given in [5]. The authors suggest that schemes are only conditionally stable for $s_p \geq 2$. To ensure unconditional stability, we therefore select $s_u = 2$ and $s_p = 1$. With these choices, the pressure-corrector scheme using the rotational formulation can be expected to be Δt^2 accurate in the L^2-norm of the velocity and $\Delta t^{3/2}$ accurate in the L^2-norm of the pressure.

3. Spatial discretization

3.1. Notation

Boilerplate

3.2. Finite element spaces

boilerplate

3.3. HDG Pressure-correction scheme

The choice of spatial discretization for the pressure gradient terms and the velocity divergence term is of central importance to the stability and robustness of the projection schemes in [4]. Previous work in [13] conducted limited investigation of the discretization of these terms as they appeared in HDG discretizations of projection methods. This motivates the present work in determining whether the discretization of these two terms is similarly important to the robustness of HDG schemes.

The point of departure for HDG schemes is to write each semi-discretized PDE as a first-order system, which we do for the velocity predictor equation (7) and pressure correction equation (11). The projection step does not require an implicit formulation and is computed directly.

3.3.1. HDG formulation of explicit operators

Pressure gradient terms. The pressure gradient term can We integrate by parts and replace p_h on the element boundary ∂K with a central numerical flux $p_h^* = \{\{p_h\}\}$. Applying the mean operator definitions on the interior and boundary interfaces separately,

$$\mathrm{pg}_h(\boldsymbol{v}, p_h, g_N^p) = -\left(\nabla\cdot\boldsymbol{v},\, p_h\right)_{\mathcal{T}_h} + \langle \boldsymbol{v}, \{\{p_h\}\}\,\boldsymbol{n}\rangle_{\partial\mathcal{T}_h\setminus\Gamma} + \langle \boldsymbol{v}, g_N^p\boldsymbol{n}\rangle_{\Gamma_N} + \langle \boldsymbol{v}, p_h\boldsymbol{n}\rangle_{\Gamma_D} \tag{16}$$

As in Fehn et al., we consider also an alternate reference formulation used in [6, 13], which does not integrate the pressure gradient by parts,

$$\mathrm{pg}_{h,\mathrm{ref}}(\boldsymbol{v}, p_h) = \left(\boldsymbol{v},\, \nabla p_h\right)_{\mathcal{T}_h} \tag{17}$$

Advection term. The advection term

$$F_a(\boldsymbol{v}, \boldsymbol{u}_h) \tag{18}$$

Velocity divergence term. The HDG formulation of the velocity divergence term pertains to the discretization of terms of the form $\left(w,\, \nabla\cdot\bar{\boldsymbol{u}}_h\right)_K$ over an element K. Just as for the pressure gradient term, we integrate by parts and replace $\boldsymbol{u}_h$ on the element boundary ∂K with a central numerical flux $\bar{\boldsymbol{u}}_h^* = \{\{\bar{\boldsymbol{u}}_h\}\}$. Applying the mean operator definitions on the interior and boundary interfaces separately,

$$\mathrm{vd}_h(w, \bar{\boldsymbol{u}}_h, \boldsymbol{g}_D^u) = (\nabla w,\, \bar{\boldsymbol{u}}_h)_{\mathcal{T}_h} + \langle w,\, \{\{\bar{\boldsymbol{u}}_h\}\}\cdot\boldsymbol{n}\rangle_{\partial\mathcal{T}_h\setminus\Gamma} + \langle w,\, \bar{\boldsymbol{u}}_h\cdot\boldsymbol{n}\rangle_{\Gamma_N} + \langle w,\, \boldsymbol{g}_D^u\cdot\boldsymbol{n}\rangle_{\Gamma_D}, \tag{19}$$

where we could have taken $\boldsymbol{g}_D^u = \bar{\boldsymbol{u}}_h$ instead, since this is enforced in equation (8). Just as for the pressure gradient term, we consider also an alternate reference formulation given in [6],

$$\mathrm{vd}_{h,\mathrm{ref}}(w, \bar{\boldsymbol{u}}_h) = \left(w, \nabla\cdot\bar{\boldsymbol{u}}_h\right)_{\mathcal{T}_h}, \tag{20}$$

which does not perform integration by parts.

note: in [13], We integrate by parts another time (equivalent) but take $\widehat{\bar{\boldsymbol{u}}}_h^{k+1}$ as the HDG flux from the predictor solve, which may be unsound.

In addition to the form of the terms themselves, HDG methods provide different choices of numerical flux which we have to investigate. Using the HDG fluxes themselves as numerical fluxes for these terms turns out to require quite complicated accounting in order to make sure the scheme stays consistent. Since the HDG flux is in some sense an intermediate quantity designed to allow for static condensation and reduction of globally-coupled degrees of freedom, it makes more sense to avoid using these fluxes elsewhere in the time discretization. Further, storing the fluxes incurs additional memory costs and requires correction [13].

3.3.2. Velocity predictor

Rewritten as a first-order system, equation (7) takes the form

$$\begin{aligned} \bar{\boldsymbol{L}} - \nabla \boldsymbol{u} &= 0 \\ \frac{\beta_s \bar{\boldsymbol{u}} - \sum_{i=0}^{s_u-1} \left(\beta_i \boldsymbol{u}^{n-i}\right)}{\Delta t} - \nabla \cdot \left(\nu \bar{\boldsymbol{L}}\right) &= -\sum_{i=0}^{s_p-1} \left(\gamma_i \nabla p^{n-i}\right) - \nabla \cdot \left(\boldsymbol{u}^n \otimes \boldsymbol{u}^n\right) + \boldsymbol{f}(t_{n+1}), \end{aligned} \tag{21}$$

where the first equation defines a new tensor-valued unknown $\bar{\boldsymbol{L}}$ approximating the velocity gradient $\nabla \bar{\boldsymbol{u}}$, and the second is equation (7) written in terms of $\bar{\boldsymbol{L}}$. Taking the numerical flux definition $\left(-\nu \hat{\bar{\boldsymbol{L}}}_h\right) \boldsymbol{n} \equiv \left(-\nu \bar{\boldsymbol{L}}_h\right) \boldsymbol{n} + \tau(\bar{\boldsymbol{u}}_h - \hat{\boldsymbol{u}}_h)$ and adding an equation to weakly enforce the continuity of its normal component on the space M_h^p [10, 11], we arrive at the following weak form

The velocity predictor $\bar{\boldsymbol{u}}_h$

$$\begin{aligned} \left(\boldsymbol{G}, \bar{\boldsymbol{L}}_h\right)_{\mathcal{T}_h} + \left(\nabla \cdot \boldsymbol{G}, \bar{\boldsymbol{u}}_h\right)_{\mathcal{T}_h} - \left\langle \boldsymbol{G} \cdot \boldsymbol{n}, \hat{\boldsymbol{u}}_h \right\rangle_{\partial \mathcal{T}_h} &= 0 \\ \left(\boldsymbol{v}, \frac{\beta_s}{\Delta t} \bar{\boldsymbol{u}}\right)_{\mathcal{T}_h} - \left(\boldsymbol{v}, \nabla \cdot \left(\nu \bar{\boldsymbol{L}}_h\right)\right)_{\mathcal{T}_h} + \left\langle \boldsymbol{v}, \tau\left(\bar{\boldsymbol{u}}_h - \hat{\boldsymbol{u}}_h\right) \right\rangle_{\partial \mathcal{T}_h} &= \left(\boldsymbol{v}, \sum_{i=0}^{s_u-1} \left(\frac{\beta_i}{\Delta t} \boldsymbol{u}^{n-i}\right)\right)_{\mathcal{T}_h} \\ - \sum_{i=0}^{s_p-1} \left(\gamma_i \, \mathrm{pg}_h(\boldsymbol{v}, p_h^{n-i}, g_N^p(t_{n-i}))\right) + F_a(\boldsymbol{v}, \boldsymbol{u}_h) + (\boldsymbol{v}, \boldsymbol{f})_{\mathcal{T}_h} \\ \left\langle \boldsymbol{\mu}, (-\nu \bar{\boldsymbol{L}}_h)\boldsymbol{n} + \tau(\bar{\boldsymbol{u}}_h - \hat{\boldsymbol{u}}_h) \right\rangle_{\partial \mathcal{T}_h \setminus \Gamma_D} + \left\langle \boldsymbol{\mu}, \widehat{b}_h \right\rangle_{\Gamma_N} &= 0 \end{aligned} \tag{22}$$

This differs from the scheme in [13], which takes the jump of the pressure in the numerical flux.

This admits a matrix-discretization over each element:

$$\begin{bmatrix} A & B & -C \\ B^T & -D & E \\ -N & G & -H \end{bmatrix} \begin{bmatrix} L_h \\ U_h \\ \widehat{U}_h \end{bmatrix} = \begin{bmatrix} 0 \\ -(F - P) \\ -L \end{bmatrix}. \tag{23}$$

In the case where the velocity components are not coupled on the boundary of the mesh due to the boundary conditions, for example, when a zero Dirichlet no-slip condition is applied to the velocity along a non-trivial bathymetry, and the vertical sides of the domain are axis-aligned, the linear system arising from equation (**??**) is decoupled, and the predictor velocity can be solved component-by-component.

Let $\bar{\phi}$ represent any component of the velocity $\bar{\boldsymbol{u}}$. If we were to write the usual form of the HDG problem where $\boldsymbol{q} = \nu \nabla \bar{\phi}$ represents the complete gradient of the component $\bar{\phi}$, then we want to find $(\bar{\phi}, \boldsymbol{q}) \in W_h \times \boldsymbol{V}_h$ such that

$$\begin{aligned} \left(\boldsymbol{v}, \nu^{-1} \boldsymbol{q}\right)_{\mathcal{T}_h} + \left(\nabla \cdot \boldsymbol{v}, \bar{\phi}\right)_{\mathcal{T}_h} - \left\langle \boldsymbol{v} \cdot \boldsymbol{n}, \hat{\phi} \right\rangle_{\partial \mathcal{T}_h} &= 0 \\ \left(w, \frac{\bar{\phi}}{a \Delta t}\right)_{\mathcal{T}_h} + \left(w, \nabla \cdot (\boldsymbol{q})\right)_{\mathcal{T}_h} - \left\langle w, \tau\left(\bar{\phi} - \hat{\phi}\right) \right\rangle_{\partial \mathcal{T}_h} &= \mathrm{rh}_h(w, \boldsymbol{u}^k, p_h^k, \boldsymbol{F})_K \\ \left\langle \mu, \widehat{\boldsymbol{q}} \cdot \boldsymbol{n} \right\rangle_{\partial \mathcal{T}_h} &= \left\langle \mu, g_N \right\rangle_{\Gamma_N} \end{aligned} \tag{24}$$

where the operator $\mathrm{rh}_h(\cdot)$ contains all the component-wise right-hand side terms of the momentum equation. This formulation is consistent with the full 3D velocity predictor if $\nu_{xy} = \nu_z$. This also couples the domain together on the interfaces. The velocity predictor $\bar{\boldsymbol{u}}_h^{k+1}$ is the vector $(\bar{\phi}_u, \bar{\phi}_v, \bar{\phi}_w)$ resulting from each of the HDG solves.

In the decoupled case, the formulation of the DG right-hand side operators can be obtained by considering only the relevant component w of the test function $\boldsymbol{v} \in \boldsymbol{V}_h^p$ as appears in equation (**??**). For the advective term, we have

$$a_h(w, \bar{\phi}, \boldsymbol{u}_h^k, \boldsymbol{g}_D) = -\left(w, \nabla \cdot \left(\bar{\phi} \boldsymbol{u}_h^k\right)\right)_K = -\left(\nabla w, \bar{\phi} \boldsymbol{u}_h^k\right)_K + \left\langle w, \left(\bar{\phi} \boldsymbol{u}_h^k\right)^* \cdot \boldsymbol{n} \right\rangle_{\partial K}, \tag{25}$$

where the operator $a_h(\cdot)$ can be be treated explicitly by taking $\bar{\phi} = u_{h,j}^k$ where the integer j denotes the spatial dimension corresponding to the component of the predictor velocity sought, or it can be treated

semi-implicitly (*cf.* [9]) as written; in the latter case, the weak form is no longer symmetric. In this paper, we consider a completely explicit treatment. need to consider boundary treatment for the advective flux–compare table in Fehn with naive approach! The numerical flux in this work $\left(\bar{\phi}\boldsymbol{u}_h^k\right)^*$ is taken to be an upwind flux. Since $\boldsymbol{u}_h^k$ is multiply-valued, either the average value $\{\{\boldsymbol{u}_h\}\}$ or the HDG flux $\hat{\boldsymbol{u}}_h$ can be used.

Similarly, the pressure gradient term for the component i of $\bar{\boldsymbol{u}}_h^{k+1}$, $\bar{\phi}_i$, is

$$\mathrm{pg}_h\left(w, p_h^k\right) = -\left(\frac{\partial w}{\partial x_i}, p_h'^{,k}\right) + \left\langle w, \left(p_h'^{,k}\right)^* n_i \right\rangle \tag{26}$$

where n_i denotes the i^{th} component of the outward unit normal $\boldsymbol{n}$, and where the numerical flux is again chosen to be the central flux, $\left(p_h'^{,k}\right)^* = \left\{\left\{p_h'^{,k}\right\}\right\}$.

3.3.3. Pressure corrector

The weak form for the pressure corrector can be expressed as

$$\begin{aligned}
(\boldsymbol{v}, \boldsymbol{q}_{\delta p}^{k+1})_{\mathcal{T}_h} + (\nabla\cdot\boldsymbol{v}, \delta p^{k+1})_{\mathcal{T}_h} - \langle \boldsymbol{v}\cdot\boldsymbol{n}, \widehat{\delta p}\rangle_{\partial\mathcal{T}_h} &= 0 \\
-(w, \nabla\cdot\boldsymbol{q}_{\delta p}^{k+1})_{\mathcal{T}_h} + \langle w, \tau_p\delta p^{k+1}\rangle_{\partial\mathcal{T}_h} - \langle w, \tau_p\widehat{\delta p}\rangle_{\partial\mathcal{T}_h} &= -\frac{\beta_s}{\Delta t}\mathrm{vd}_h(w, \bar{\boldsymbol{u}}_h, \boldsymbol{g}_D^u(t_{n+1})) \\
\left\langle \mu, \boldsymbol{q}_{\delta p}^{k+1}\cdot\boldsymbol{n} + \tau_p(\delta p^{k+1} - \widehat{\delta p})\right\rangle_{\partial\mathcal{T}_h\setminus\Gamma_D} &= 0
\end{aligned} \tag{27}$$

3.4. Inhomogeneous boundary condition treatment

3.5. Numerical flux definitions

4. Implementation

The finite element forward models were implemented in `C++` and make use of the finite element library `deal.ii` [**?**].

All surface and volume integral operators are discretized with Gaussian quadrature using $p_{\text{order}} + 1$ one-dimensional (1D) quadrature points in each spatial direction, where p_{order} is the polynomial order of the finite element space (space).

Convergence results are reported using the relative L^2-errors

$$\mathrm{e}_{\boldsymbol{u}_h}(t) = \frac{\|\boldsymbol{u}(\boldsymbol{x},t) - \boldsymbol{u}_h(\boldsymbol{x},t)\|_{L^2(\Omega_h)}}{\|\boldsymbol{u}(\boldsymbol{x},t)\|_{L^2(\Omega_h)}}, \qquad \mathrm{e}_{p_h}(t) = \frac{\|p(\boldsymbol{x},t) - p_h(\boldsymbol{x},t)\|_{L^2(\Omega_h)}}{\|p(\boldsymbol{x},t)\|_{L^2(\Omega_h)}}, \tag{28}$$

where each of the L^2-norms over the computational domain Ω_h is calculated using Gaussian quadrature over the element volumes using $p_{\boldsymbol{u}} + 3$ for errors in the velocity and $p_p + 3$ for errors in the pressure variables. Unless explicitly specified, the abbreviations $\mathrm{e}_{\boldsymbol{u}_h}$ and e_{p_h} refer to the velocity and pressure errors at the final time T.

4.1. Computational considerations

HDG methods offer several well-known advantages over standard DG-FEM methods. In addition to favorable convergence properties, hybridization of the system by introducing the space Mph can lead to a substantial reduction in globally-coupled degrees of freedom as compared to classical DG methods. However, the complexities associated with the static condensation procedure introduce a set of non-typical computational performance considerations as compared to other high-order methods. In this section, we provide a brief discussion of practical computational considerations specific to HDG schemes in the context of parallelization and matrix-free solvers. These topics are often overlooked or unaddressed in the literature (to DG/HDG etc ref).

4.2. Parallelization

A commonly cited performance advantage of HDG is the embarrassingly parallel nature of the assembly as well as the local reconstruction of the numerical solution QH UH from the trace quantity UHAT. Means that people brush this step off computationally, as a parallel implementation should make it trivial. However there's more to the story than that.

- argument: that assembly and reconstruction start to matter at large problem sizes
- parallelization needed to beat down assembly and reconstruction time
- multi-threaded parallel applications highly dependent on polynomial order

though it may seem counterintuitive, multi-threaded programs can sometimes run slower than their single-threaded or serial counterparts due to several reasons:

Overhead: Threads require resources for creation, termination, and synchronization. The overhead of managing multiple threads can slow down the program, especially if the computation performed by each thread is not significant compared to this overhead. Contention: When multiple threads try to access a shared resource simultaneously, contention can occur. This could lead to a situation called a lock, where one thread has to wait for another to release a resource. In extreme cases, this can lead to a slowdown known as lock contention, where the threads spend more time waiting to access the resource than doing useful work. False sharing: Even if different threads are working on different data, if that data is close enough in memory (i.e., in the same cache line), then the hardware can treat it as if it were shared. When one thread updates its data, the hardware will think the entire cache line has been modified and will have to update it across all the cores, leading to a significant slowdown. This is known as false sharing. Load imbalance: If the work is not evenly distributed across threads, some threads may finish their work early and sit idle, while others are still working. This is known as load imbalance. It can result in the execution time of the program being dominated by the slowest thread. Non-parallelizable tasks: Some tasks simply can't be parallelized effectively due to their nature, they are inherently serial (or "sequential"). This principle is encapsulated by Amdahl's Law, which states that the maximum improvement to a system's performance is limited by the portion of the system that can't be parallelized. Memory limits: Every thread created uses some memory for its stack. If the number of threads is too high, it can cause significant memory usage, potentially leading to swapping if the system runs out of physical memory, which can drastically decrease performance. It's worth noting that developing an efficient multi-threaded program requires careful design to minimize these issues. This can involve techniques such as fine-tuning the number of threads, avoiding contention and false sharing, balancing the load properly, and choosing appropriate data structures and synchronization primitives. But even with all these considerations, there's no guarantee that a multi-threaded solution will always outperform a well-optimized single-threaded one. It highly depends on the nature of the problem and the specific hardware the software is running on.

In this case the synchronization work necessary to ensure no race condition while copying elemental contributions into the global linear system is highly sensitive to problem dimension and polynomial order. The necessary work on the cell

It turns out that for one-dimensional problems using any polynomial order less than or equal to $p = 10$, it is far more efficient to perform assembly in serial as opposed to in parallel using multi-threading. This is because the problem dimension leads to element-local systems small enough that the synchronization work between the threads to prevent a race condition copying into the global linear system is much larger than the actual work on the cell.

4.3. Matrix-free solvers

4.4. On the solution of the pure Neumann problem

In the case of pure Dirichlet boundaries on the velocity predictor, the pressure level is undefined. To see this, note that if the scalar field δp_h is a solution of (9) with $\Gamma_N = \partial\Omega$, then $\delta p_h + c$ for any $c \in \mathbf{R}$ is also a solution. Discretely, the null space of the linear system arising due to the discretization of (9) is spanned by the set of constant vectors, implying a singular linear system.

Although crucial to the performance of pressure correction schemes, the issue of finding a solution to the singular system in such cases receives little attention in the literature. iterative solvers [1, 7], An approach

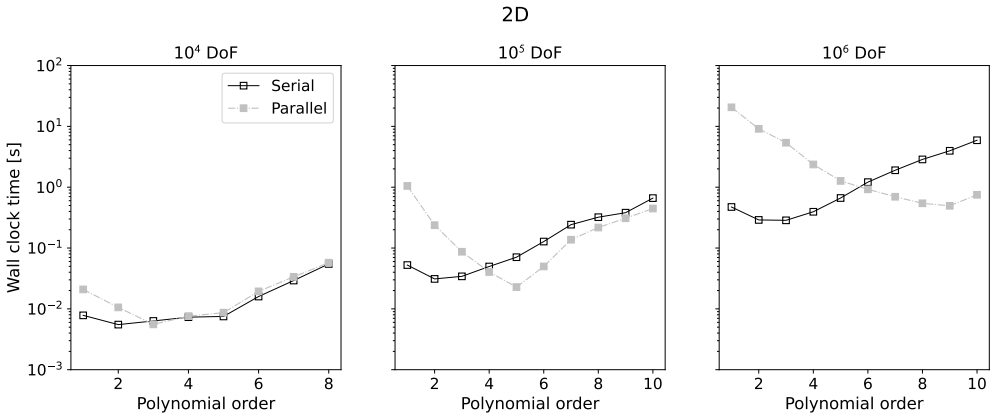

Figure 1: Wall clock times for 2D HDG assembly at constant problem size, serial vs. parallel

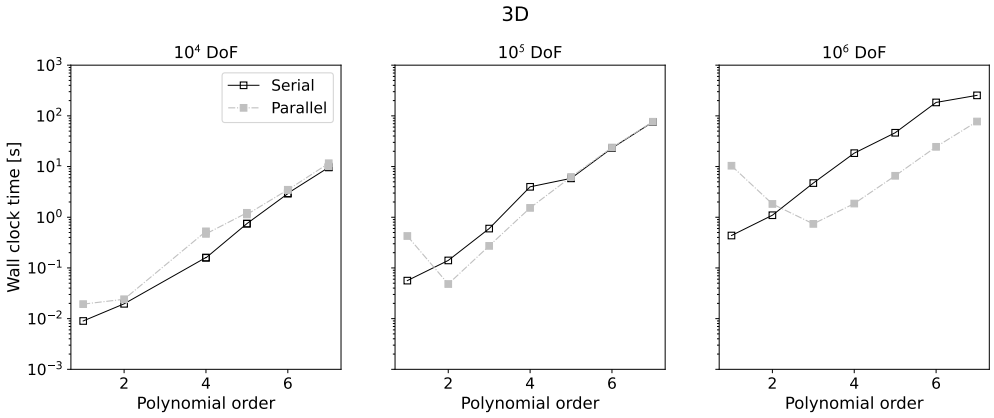

Figure 2: Wall clock times for 3D HDG assembly at constant problem sizes, serial vs. parallel

favored by many practitioners is to manually specify the value of the candidate solution at a single point by removing an equation from the discrete system and applying a Dirichlet constraint fixing the value of the solution at that point to an arbitrary constant, eliminating the null space and allowing solution of the linear system using a conventional direct solver. However, in a finite-element context, the function δ_p is often represented in the Sobolev space $H^1(\Omega)$ or $L^2(\Omega)$, spaces in which point evaluations do not make sense. In these cases, imposing such a constraint can render the variational problem ill-posed.

This is complicated by the fact that some numerical linear algebra software varies widely in terms of implementation; while some libraries will not solve a singular system (numpy), other direct procedures will do so when they can identify a zero-pivot. Similar for iterative solvers.

In [2], the authors rigorously describe strategies for addressing the singular system in a continuous finite element context. Here, we extend that discussion to the DG-FEM setting and illustrate computational trade-offs associated with some candidate approaches.

As a first approach, we apply a subspace projection using a Krylov-solver [14], making use of the fact that iterative solvers solve singular systems provided that the right-hand side is in the orthogonal complement of the null-space. This modification is ok because we're only removing the null space contribution to the linear system As a second approach, we apply a penalty-based method which can be interpreted as a regularization. This method, while simple and independent of linear solver type, involves the specification of a hyperparameter γ. As a third approach, we impose a mean-value constraint

$$\int_\Omega \delta p \, d\Omega = 0 \tag{29}$$

into the linear system directly, avoiding the saddle point system that arises as a result of applying the constraint as a Lagrange multiplier [2].

As a test case, we use a manufactured solution How to do the zero-mean removal?

Boundary conditions and forcing function are deduced from the exact solution

$$\delta_p^* = \frac{1}{3} \sin\left(-\frac{\pi}{2}x\right) \cos\left(2\pi y\right). \tag{30}$$

We consider the domain $\Omega = [-1, 1]^2$, and by symmetry, we have that the exact solution δ_p^* is analytically zero-mean in the sense of equation (29).

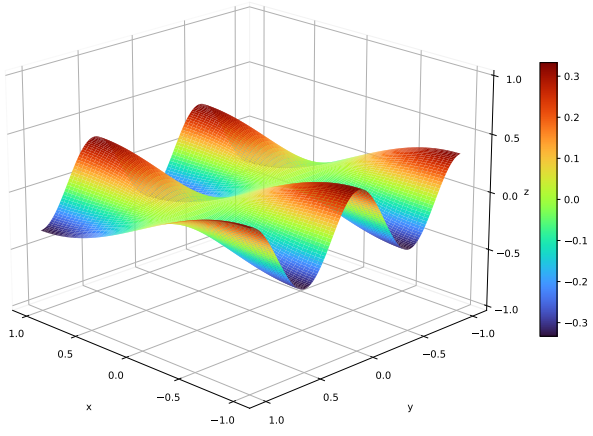

Figure 3: Analytical solution of the pure Neumann problem

5. Numerical experiments

5.1. Unsteady Stokes Flow

The review paper [5] solves the unsteady Stokes problem

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x},t) &= \pi \sin t \begin{pmatrix} \sin(2\pi y)\sin^2(\pi x) \\ -\sin(2\pi x)\sin^2(\pi y) \end{pmatrix} \\ p(\boldsymbol{x},t) &= \sin(t)\cos(\pi x)\sin(\pi y) \end{aligned} \tag{31}$$

where the source term $f = u_t - \nu\nabla\cdot(\nabla\boldsymbol{u}) + \nabla p$.

In [3], we have

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x},t) &= \begin{pmatrix} \sin(x)(a\sin(ay) - \cos(a)\sinh(y)) \\ \cos(x)(\cos(ay) + \cos(a)\cosh(y)) \end{pmatrix} \exp(-\lambda t), \\ p(\boldsymbol{x},t) &= \lambda\cos(a)\cos(x)\sinh(y)\exp(-\lambda t) \end{aligned} \tag{32}$$

where $\lambda = \nu(1+a^2)$ $\nu = 1$ and $a = 2.883356$ on a domain of $\Omega = [-1,1]^2$; they take $[0,T] = [0,0.1]$ and Dirichlet boundary conditions everywhere, $\Gamma = \Gamma_D$.

5.2. Verification

Spatial convergence test, temporal convergence test

5.3. Taylor-Green vortex flow

Is this the same test case as in Hesthaven?

5.4. Backward facing step

5.5. Lid-driven cavity flow

5.6. Lock Exchange?

5.7. Lab RTI flow?

6. Discussion

The imposition of open boundary conditions where inflow and outflow can co-exist remains an open problem in the finite element community [12].

7. Appendix: BDF1

References

[1] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Mar. 1996. Google-Books-ID: hNpJg_pUsOwC.

[2] P. Bochev and R. B. Lehoucq. On the Finite Element Solution of the Pure Neumann Problem. *SIAM Review*, 47(1):50–66, Jan. 2005. Publisher: Society for Industrial and Applied Mathematics.

[3] N. Fehn, W. A. Wall, and M. Kronbichler. On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations. *Journal of Computational Physics*, 351:392–421, Dec. 2017.

[4] N. Fehn, W. A. Wall, and M. Kronbichler. Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows. *Journal of Computational Physics*, 372:667–693, Nov. 2018.

[5] J. L. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195(44):6011–6045, Sept. 2006.

[6] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods*, volume 54 of *Texts in Applied Mathematics*. Springer New York, New York, NY, 2008.

[7] R. Iankov, M. Datcheva, S. Cherneva, and D. Stoychev. *Finite Element Simulation of Nanoindentation Process*. Jan. 2013.

[8] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97(2):414–443, Dec. 1991.

[9] N. C. Nguyen, J. Peraire, and B. Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for linear convection–diffusion equations. *Journal of Computational Physics*, 228(9):3232–3254, May 2009.

[10] N. C. Nguyen, J. Peraire, and B. Cockburn. A hybridizable discontinuous Galerkin method for Stokes flow. *Computer Methods in Applied Mechanics and Engineering*, 199(9):582–597, Jan. 2010.

[11] N. C. Nguyen, J. Peraire, and B. Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 230(4):1147–1170, Feb. 2011.

[12] R. L. Sani and P. M. Gresho. Résumé and remarks on the open boundary condition minisymposium. *International Journal for Numerical Methods in Fluids*, 18(10):983–1008, 1994. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650181006.

[13] M. P. Ueckermann and P. F. J. Lermusiaux. Hybridizable discontinuous Galerkin projection methods for Navier–Stokes and Boussinesq equations. *Journal of Computational Physics*, 306:390–421, 2016.

[14] H. A. v. d. Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Apr. 2003. Google-Books-ID: wE0NrHkrqRAC.