Archives of Computational Methods in Engineering
https://doi.org/10.1007/s11831-020-09502-5

ORIGINAL PAPER

HDGlab: An Open-Source Implementation of the Hybridisable Discontinuous Galerkin Method in MATLAB

Matteo Giacomini[1,2] · **Ruben Sevilla**[3] · **Antonio Huerta**[1,2]

Received: 10 September 2020 / Accepted: 24 September 2020
© The Author(s) 2020

Abstract
This paper presents HDGlab, an open source MATLAB implementation of the hybridisable discontinuous Galerkin (HDG) method. The main goal is to provide a detailed description of both the HDG method for elliptic problems and its implementation available in HDGlab. Ultimately, this is expected to make this relatively new advanced discretisation method more accessible to the computational engineering community. HDGlab presents some features not available in other implementations of the HDG method that can be found in the free domain. First, it implements high-order polynomial shape functions up to degree nine, with both equally-spaced and Fekete nodal distributions. Second, it supports curved isoparametric simplicial elements in two and three dimensions. Third, it supports non-uniform degree polynomial approximations and it provides a flexible structure to devise degree adaptivity strategies. Finally, an interface with the open-source high-order mesh generator Gmsh is provided to facilitate its application to practical engineering problems.

Keywords Hybridizable discontinuous Galerkin · High-order · Elliptic problems · MATLAB · Open-source

Mathematics Subject Classification 35-04 · 35M32 · 65N30 · 68-04 · 97N80

1 Introduction

In recent years, hybrid discretisation methods have received increasing attention by the applied mathematics and computational engineering community. The main interest in these methodologies is due to their reduced computational cost with respect to classical discontinuous Galerkin (DG) methods, see [135, 163, 170, 267], from which they inherit appealing stability and convergence properties as well as the flexibility to devise high-order, non-uniform degree and adaptive discretisations and the capability to efficiently exploit parallel computing architectures [42, 91, 108, 158, 226].

The purpose of the present contribution is two-fold: to present a review on the state-of-the-art of hybrid discretisation methods including both fundamental and applied contributions; to provide an educational implementation of the hybridisable discontinuous Galerkin (HDG) method in MATLAB, the so-called HDGlab library, and describe its structure, capabilities and functioning. HDGlab is an open-source library released under GNU GPL licence and designed for rapid prototyping and testing. It supports simplicial meshes and it provides a seamless 2D and 3D implementation with vectorised loops on the integration points. In addition, HDGlab presents four specific features, currently not available in existing open-source HDG implementations in MATLAB:

1. Availability of high-order polynomial shape functions up to degree 9, with both equally-spaced and Fekete nodal distributions.
2. Support of curved isoparametric simplicial elements in 2D and 3D.

✉ Ruben Sevilla
r.sevilla@swansea.ac.uk

[1] Laboratori de Calcul Numeric (LaCàN), ETS de Ingenieros de Caminos, Canales y Puertos, Universitat Politècnica de Catalunya, Barcelona, Spain

[2] International Centre for Numerical Methods in Engineering (CIMNE), Barcelona, Spain

[3] Zienkiewicz Centre for Computational Engineering, College of Engineering, Swansea University, Bay Campus, Swansea SA1 8EN, Wales, UK

Published online: 22 October 2020

3. Support of non-uniform degree polynomial approximations and flexibility to devise degree adaptivity strategy.
4. Interface with the open-source high-order mesh generator `Gmsh`.

The remainder of this paper is organised as follows. First, a review of the state-of-the-art on hybrid discretisation methods is presented in Sect. 2. The formulation of the HDG method for the Poisson and Stokes problems is briefly recalled in Sects. 3 and 4, respectively. Section 5 provides a description of the structure of the `HDGlab` library and the url of the repository available under GNU GPL licence. The data structures for the storage of the mesh information, the reference element and the reference face are presented in Sect. 6. Section 7 is devoted to the preprocessing operations, whereas the core of the `HDGlab` solver for the scalar Poisson equation is described in Sect. 8. Its extension to vectorial problems involving incompressible Stokes flows is discussed in Sect. 9. The visualisation library is introduced in Sect. 10. Section 11 is devoted to numerical examples, in 2D and 3D, validating the optimal convergence properties of the HDG method and showing the potentialities of the `HDGlab` implementation. Finally, Sect. 12 summarises the capabilities of the presented library and three appendices provide implementation details for the Poisson (Appendix A) and Stokes (Appendix B) solvers and for the interface with the mesh generator `Gmsh` (Appendix C).

2 Literature Review

The common idea of all hybrid discretisation methods stems from the seminal works of Guyan on static condensation of primal formulations [157] and of Fraeijs de Veubeke on hybridisation of mixed formulations [138] of the finite element method. In the context of element-by-element discontinuous approximations, these techniques allow to remedy the drawback of node duplication in DG methods by considering only the unknowns on the mesh faces (edges in 2D) as globally-coupled degrees of freedom. More precisely, the unknowns in each element are expressed as a function of the degrees of freedom on the element faces by solving a local boundary value problem with purely Dirichlet data, whereas appropriate transmission conditions are imposed to guarantee the interelement continuity of the solution and the fluxes, see [75].

Three families of hybrid numerical schemes lay within this description, namely, (1) hybrid/hybridised DG, (2) hybridisable DG, henceforth referred to as HDG, and (3) hybrid high order (HHO) methods. Stemming from classical DG primal formulations, the hybrid or hybridised DG method reduces the number of globally coupled degrees of freedom by performing static condensation [124–126]. In addition, improved efficiency can be achieved using polynomial spaces of degree $p+1$ and p for the primal and hybrid variables, respectively and resorting to the reduced stabilisation approach [206, 207]. The hybridisable DG method, henceforth named HDG, is derived from the mixed formulation of the local DG method [77, 87, 99] with hybridisation. The main advantage of HDG with respect to other hybridised DG methods relies in the introduction of a mixed variable approximating the gradient of the primal unknown [88, 89]. This approach is of special interest in the context of engineering problems where quantities of interest often depend on the flux of the solution or on the stress. Finally, HHO bridges the two approaches above by utilising a primal formulation and introducing a local reconstruction operator for the gradient of the solution and an appropriate stabilisation term in the static condensation problem [109, 110]. It is worth noting that many hybrid discretisation schemes can be interpreted in a unique framework as HDG-type methods via appropriate definitions of the stabilisation term, see e.g. [68, 69] for the staggered DG method and [106] for HHO.

Unified presentations of hybrid discretisation techniques and their relationship with other known numerical methods are available in [37, 89, 114]. Interested readers are also referred to the review papers [75, 149] and to the recent monograph [112]. In the following subsections, an overview of the contributions on hybrid discretisation methods according to the authors' vision is presented.

2.1 From Linear to Nonlinear Scalar Equations

Second-order scalar elliptic problems have been extensively studied using HDG [89], HHO [110, 115] and the hybridised DG method [206], whereas their extension to linear convection–diffusion problems is discussed in [79, 113, 124, 200]. Cases of higher-order partial differential equations (PDEs) are presented in [78] and [62] for HDG discretisations of biharmonic and third-order equations, respectively, whereas an HHO approximation of the Cahn-Hilliard equation is proposed in [53]. In addition, time-fractional diffusion problems are discussed in [76, 197].

More recently, there has been growing interest towards the analysis and simulation of quasilinear and semilinear problems, including the quasilinear p-Laplace operator [95, 216] and the semilinear Grad–Shafranov equation [233, 234]. To reduce the computational cost of semilinear problems, the interpolatory HDG method was recently devised introducing an interpolation procedure for the efficient and accurate approximation of nonlinear terms [54, 100].

Concerning nonlinear problems, HDG discretisations were proposed for nonlinear convection–diffusion [201] and nonlinear Schrödinger [46] equations, whereas an HHO formulation of the nonlinear Leray–Lions equation is presented in [111]. Recent applications involving HDG approximations

of nonlinear scalar equations focus on the optoelectronic simulation of photovoltaic solar cells. This problem couples a high-order HDG method for the drift-diffusion electronic model in the semiconductor layer of the solar cells with an efficient approximation of the time-harmonic Maxwell's equations [21, 56].

2.2 Incompressible Flows

In the context of incompressible flows, HDG formulations of the Stokes equations were devised and analysed in [90, 92, 98, 199]. The corresponding analysis of the HDG method for Oseen flow is presented in [49]. In [92], it was observed that the HDG method based on Cauchy stress tensor formulation experiences suboptimal convergence of the mixed variable and loss of superconvergence of the postprocessed velocity, when low-order polynomial approximations are considered. The M-decomposition approach [84] remedies this issue by appropriately enriching the discrete local spaces of approximation. An alternative strategy imposing the symmetry of the mixed variable pointwise via Voigt notation is discussed in [148] along with a postprocessing procedure to handle translational and rotational rigid body modes. Divergence-conforming HDG [94], hybridised DG [125] and embedded-hybridised DG (EHDG) [225] discretisations were also studied for the incompressible Stokes equations, whereas a pressure-robust HHO method for viscosity-dependent Stokes flows is proposed in [116].

HDG formulations for the nonlinear incompressible Navier–Stokes equations using equal order and different order of polynomial approximations for the primal, mixed and hybrid variables are described in [50, 204] and [215], respectively. The former approach is also employed in [152] to devise a degree adaptive strategy relying on the local superconvergence of the postprocessed velocity. Stemming from the work in [117], different HHO formulations of the incompressible Navier–Stokes equations were proposed, incorporating a skew-symmetric form of the convection term [31] and a globally divergence-free velocity approximation to achieve robustness in presence of large irrotational body forces [45]. Moreover, special attention was devoted in recent years to the development of hybridised DG schemes [126] with pointwise divergence-free velocity [171, 181, 223] and with relaxed H(div)-conformity [177, 178], as well as divergence-conforming hybrid DG discretisations for incompressible flows on surfaces [179]. It is worth noting that all the above mentioned references focus on viscous laminar flows and preliminary promising results on the incompressible Reynolds averaged Navier–Stokes (RANS) equations coupled with the Spalart-Allmaras turbulence model were recently presented in [210].

Besides classical approaches to steady and unsteady Navier–Stokes equations, HDG-based space-time formulations were studied for their ability to effectively handle moving and deforming domains. More precisely, stemming from the HDG formulation introduced in [222], H(div)-conforming hybridised DG [160] and EHDG [161] methods were proposed. Hybridised DG and HDG methods with arbitrary Lagrangian Eulerian (ALE) formulations were thus presented in [134, 140] and the resulting HDG-ALE framework was applied to fluid–structure interaction (FSI) problems involving incompressible [248] and weakly-compressible flows [176].

Among the applications of hybrid discretisation methods to incompressible flows, it is also worth mentioning the recent attempts to simulate quasi-Newtonian fluids [145] and viscoplastic materials [44].

2.3 Two-Phase Flows and Heterogeneous Porous Media

HDG simulations of immiscible incompressible two-phase flows in heterogeneous porous media were first proposed in [130] and coupled with high-order diagonally implicit Runge–Kutta (DIRK) time integrators in [107]. Moreover, in [168] a linear degenerate elliptic problem modelling two-phase mixture is approximated using a hybridised DG approach. Darcy flow and two-phase flow simulations in highly heterogeneous media are performed in [270] via the so-called generalised multiscale HDG (GMsHDG) method which is connected to the mortar mixed finite element method described in [24]. GMsHDG was also employed for multiscale simulations of elliptic PDEs in heterogeneous media [70, 123] and perforated domains [65] and of parabolic PDEs in heterogeneous media [192].

An alternative to GMsHDG is the HHO framework for highly oscillatory elliptic problems introduced in [73]. Moreover, in the context of coupled problems involving porous media, HHO simulations of passive transport of a solute in a fractured medium are presented in [52], whereas nonlinear poroelastic phenomena in a saturated porous medium with a slightly compressible fluid are described in [33, 35].

Extensive research has been also devoted to coupled Stokes/Darcy and Brinkman models. In [51], an EHDG formulation of the Stokes/Darcy system is described. Concerning the Brinkman model, an analysis of its HDG approximation is presented in [22], its simulation in the context of heterogeneous media with high-contrast is discussed in [183] and an H(div)-conforming discretisation is proposed in [142]. In [30], an HHO formulation with divergence-conforming Darcy velocity and higher-order Stokes velocity is devised.

2.4 Compressible Flows and Gas Kinetics Equations

Hybrid formulations for inviscid Euler and laminar compressible Navier–Stokes equations are proposed in [209] in the context of HDG and in [205] for the embedded DG (EDG) method. Extension to viscous turbulent compressible flows using RANS equations with Spalart–Allmaras turbulence model is presented in [193], whereas a large-eddy simulation framework is introduced in [133]. In addition, an entropy-stable space-time discretisation was proposed for the compressible Navier–Stokes equations using an HDG approach in space and a discontinuous approximation in time [266]. More recently, special attention was dedicated to the development of positivity-preserving Riemann solvers in the context of hybridised DG methods [263]. For a complete review on HDG methods for compressible flows, interested readers are referred to [263], whereas the application to gas kinetics modelled by means of the linearised Bhatnagar–Gross–Krook equation is discussed in [254].

2.5 Plasma Physics and Magnetohydrodynamics

Computational physics community is showing increasing interest towards the application of hybrid discretisation methods to the simulation of magnetic plasma physics. Promising preliminary results concerning the HDG approximation of the Grad–Shafranov equation in axisymmetric confinement devices modelling fusion reactors are described in [233, 234]. In the context of magnetohydrodynamics (MHD), an HDG method for steady-state linearised incompressible MHD equations is proposed in [180]. Approximation strategies for the unsteady compressible MHD equations using HDG, EDG and the interior embedded DG (IEDG) methods with DIRK time integrators are explored in [74].

2.6 Shallow Water Equations

The shallow water equations have been extensively studied in the context of hybrid DG methods, starting from the linearised shallow water system in [38] to the nonlinear Korteweg-de Vries equation in [63, 231]. In both the above mentioned works, time integration is performed implicitly using a backward Euler method. Extension to high-order backward differentiation formulas is discussed in [128] in the context of the Benjamin–Bona–Mahony equation. To reduce the computational cost of fully-implicit procedures, in [228] an operator splitting is applied to the Green–Naghdi equation and the nonlinear hyperbolic subproblem is solved using an explicit approach, whereas the implicit time integrator is only applied to the linear dispersive subproblem. A similar idea is presented in [169] to devise an implicit–explicit (IMEX) HDG-DG scheme in which the linear part of the problem is solved using a hybridised DG method and a singly diagonally implicit Runge–Kutta (SDIRK) scheme and the nonlinear one is approximated by means of an explicit Runge–Kutta (RK) DG discretisation. A detailed comparison of explicit and implicit approaches to the nonlinear shallow water equations is provided in [229].

2.7 Wave Propagation Phenomena

The benefits of high-order methods in the simulation of wave propagation prompted extensive research on hybrid discretisation methods in the fields of electromagnetics, elastodynamics and acoustics. A detailed review on HDG and EDG approaches for these problems is available in [132].

Starting from the work in [203], research on time-harmonic Maxwell's equations tackled the analysis and development of HDG formulations [186], including methods suitable for simulations at large wave numbers [189] and Schwarz-type domain decomposition (DD) strategies designed for HDG [18, 185]. Recent applications of HDG to time-harmonic Maxwell's equations focus on wave propagation in heterogeneous media modelling photovoltaic cells [41], coupling with nonlocal hydrodynamic Drude and generalised nonlocal optical response models [184] and with hydrodynamic models for metals [257–259, 271] to simulate plasmonic nanostructures. In the context of time-domain Maxwell's equations, HDG methods are presented and analysed in [55, 59, 122], whereas implicit hybridised DG discretisations are proposed in [67].

In the framework of elastodynamics, HDG with DIRK time integrators were introduced in [202], whereas in [256] an HDG spectral element method (HDG-SEM) is utilised to simulate wave propagation in coupled elastic-acoustic media. In the frequency-domain, HDG methods for elastodynamics are analysed and presented in [28, 164].

The first HDG solver for acoustics, introduced in [202], relied on a fully-implicit approach based on DIRK time integrators. Since then, explicit HDG formulations utilising strong stability-preserving RK (SSPRK) and explicit RK integrators were proposed in [252], wheras an explicit arbitrary derivative (ADER) approach is discussed in [236]. In addition, a comparison of implicit and explicit HDG schemes for acoustic wave propagation is performed in [173]. More recently, an HDG-based cut finite element startegy with local time stepping was presented in [237]. It is worth recalling that devising a conservative numerical scheme is a critical aspect for the accurate simulation of acoustic wave propagation. To correct the dissipative nature of the method analysed in [93], an energy-conservative HDG formulation with a two-step Stormer-Numerov time-marching is proposed in [86]. Moreover, symplectic [232] and multisymplectic [190] HDG schemes preserving the Hamiltonian structure of the PDEs under analysis were developed to achieve energy conservation.

Among the applications of HDG to wave propagation phenomena, it is also worth mentioning the degree adaptive approximation of the mild slope equation to perform harbour simulations [152] and the cardiac electrophysiology simulations of the monodomain model [159, 227].

2.8 Linear and Nonlinear Elasticity

In linear elasticity, the imposition of the symmetry of the stress tensor using HDG methods based on mixed formulations has been extensively studied in the literature. Indeed, the first formulations introduced in [141, 251] experienced suboptimal convergence of the mixed variable and a loss of superconvergence of the postprocessed displacement field. To remedy this issue, a formulation considering a weakly symmetric stress tensor was presented in [97]. The strong imposition of the symmetry can be achieved via several strategies: in [214], different degrees of polynomial approximation are considered for the primal and hybrid variables; the M-decomposition framework [82, 83, 85] is applied to the linear elastic problem [81] to enrich the discrete spaces of approximation utilised in the local problem; an alternative formulation imposing the symmetry of the mixed variable pointwise via Voigt notation is proposed for high-order and the lowest-order HDG discretisations in [245] and [244], respectively. In the context of the high-order discretisation, a novel postprocessing strategy accounting for rigid translation and rotation is also devised. It is worth noting that hybrid methods relying on primal formulations do not suffer from these issues, see e.g. HHO [109].

Timoshenko beams are discussed in [47, 48], whereas the case of Kirchhoff plates is considered in [162] using HDG and in [27] using HHO methods.

In the context of nonlinear elasticity, the first hybrid discretisation formulation was presented in [167]. In this work, it was observed that the method may not converge to the exact solution if the interelement jumps are not appropriately penalised and a detailed numerical study on the choice of the HDG stabilisation is discussed in [96]. More recently, a locking-free HDG formulation for nonlinear elasticity of thin structures subject to large deformations was proposed [255]. In addition, HHO discretisations of hyperelastic materials in small and finite deformations were presented in [34] and [15], respectively. HHO discretisations for problems involving plastic and elastoplatic simulations are discussed in [16, 17], whereas contact phenomena are addressed in [66].

2.9 Interface Problems and Immersed Discretisations

The first attempt to solve interface problems using hybrid discretisation techniques was proposed in [165] using a body-fitted mesh. In this context, a superparametric HDG formulation was considered to limit the geometric error due to the polygonal approximation of curved interfaces.

Recently, immersed methods have received special attention, both in the context of HHO and HDG formulations. More precisely, unfitted HHO methods relying on a cell agglomeration procedure to remedy small cut instabilities are analysed for scalar and vectorial second-order elliptic problems in [39, 40]. In the framework of HDG, Poisson interface problems are treated in [121] by means of an unfitted method introducing appropriately defined ansatz functions in the vicinity of the interface. An alternative approach to handle curved interfaces is proposed in [217] where a fictitious domain strategy is developed coupling a mesh of planar faces and a transferring function for the imposition of the transmission conditions on the fictitious subdomain. Inspired by the cut finite element method, in [237], a high-order HDG strategy employing a level-set function to describe the immersed interfaces and a cell agglomeration procedure is described for the wave equation. Similarly, the extended HDG (X-HDG) method introduces a framework in which the HDG local problem is modified only in the elements cut by the interface. In this context, cut instabilities are handled by displacing the mesh nodes responsible for the bad cuts [154–156]. Finally, an HDG-based phase-field model for brittle fracture was recently proposed in [194].

2.10 High-Order and Exact Geometry Representations

Geometry representation plays a crucial role in the capability of high-order methods to achieve optimal accuracy. In the context of HDG, high-order isoparametric approaches in presence of curved meshes are utilised in many references, see e.g. [148, 193], whereas this technique is addressed for HHO in [29]. An alternative approach relying on meshes with planar faces and the extension to a fictitious subdomain is discussed in [101, 102, 250] for several linear problems and was recently extended to the semilinear Grad-Shafranov equation [233, 234]. It is worth noting that all the techniques mentioned above introduce geometric errors due to the polynomial approximation of the boundaries. In order to exploit the exact CAD representation of the boundaries, the NURBS-enhanced finite element method (NEFEM) [241, 242] is employed in [149, 239, 247] to devise HDG formulations with exact geometry for Stokes, linear elastic and electrostatics problems.

2.11 Lowest-Order Hybrid Discretisations

Hybrid discretisation methods have been traditionally developed in the context of high-order approximations. Nonetheless, it is well-known that lowest-order discretisations, e.g. the finite volume (FV) method, are more robust than

high-order techniques. In this framework, a new class of lowest-order hybrid discretisations was developed, with unknowns approximated by means of constant functions on the mesh faces. The recently proposed face-centred finite volume (FCFV) for Poisson, Stokes [243] and linear elasticity [244] can be interpreted as an HDG method of degree zero. Variants of this approach achieving optimal second-order convergence of the primal variable are discussed in [150, 262]. Stemming from HHO, lowest-order nonconforming discretisations are proposed in [32] for linear elasticity and in [72] for elliptic obstacle problems. As their high-order counterparts, the above mentioned methodologies allow the use of generic polygonal and polyhedral elements and provide a workaround to the sensitivity issues of FV methods to mesh distortion and stretching [118, 119].

2.12 Iterative Solvers and Preconditioning

Although hybrid discretisation methods are responsible for a substantial reduction of degrees of freedom with respect to classical DG methods, their applicability to realistic problems of engineering interest still rely on the development of efficient solution strategies for large-scale systems.

In [26], a DD strategy based on restricted additive Schwarz methods is proposed for hybridised DG approximations, whereas an optimised Schwarz DD approach suitable to handle the many-subdomain case is discussed in [144]. Starting from [80], several works also explored the capabilities of multigrid solvers for HDG formulations, including hierarchical scale separation [238], geometric multigrid [265], nested geometric multigrid on many-core processors [129], p-multigrid in the context of second-order elliptic problems [174] and compressible Navier–Stokes flows [139] and GPU-accelerated p-multigrid for linear elasticity [127]. Finally, iterative algorithms inspired by the Gauss-Seidel method were proposed in [196] and tested on massively parallel architectures up to 16,384 cores. A block symmetric Gauss-Seidel type preconditioner was also introduced in [224], whereas a multilevel solver coupled with a block-Jacobi fine scale solver is proposed in [195].

2.13 A Posteriori Error Estimates and Adaptivity

The quality of hybrid discretisation methods has been assessed in several works by means of a posteriori estimates of the error in the primal, mixed and hybrid variables, as well as in quantities of interest.

Starting from the seminal works [104, 105] establishing reliability and efficiency of error estimates for the HDG approximations of second-order elliptic equations, a posteriori estimates were developed for steady and unsteady scalar convection–diffusion problems [57, 182] and for the vectorial case of incompressible Oseen [23] and Brinkman [22] flows. In addition, constant-free computable a posteriori error estimates are devised in [19] for second-order elliptic problems using an equilibrated fluxes approach, whereas residual-based estimates are established for Maxwell's equations in [58].

In the context of adaptivity, on the one hand, the analysis of HDG approximations based on non-uniform polynomial degrees [60, 61] and the superconvergence property of the postprocessed solution [77, 89] prompted the development of degree adaptive procedures based on superparametric HDG methods [152, 153] and on isoparametric HDG-NEFEM approaches [149, 239, 247]. Degree adaptivity is also applied to the simulation of cardiac electrophysiology in [159]. On the other hand, mesh adaptivity procedures to capture localised abrupt changes in the solution were devised in [234] and [194] for the Grad-Shafranov equation and the phase-field model for brittle fracture, respectively. Octree-based mesh refinement is performed in [230] for anisotropic inhomogeneous diffusion problems. Mesh adaptivity driven by local error indicators is also employed in the context of second-order FCFV approximations [150, 262]. Concerning the error in quantities of interest, an adjoint-based method allowing to achieve superconvergent approximations of linear functionals is described in [103] and goal-oriented mesh adaptation strategies are proposed in [136, 268].

2.14 Coupling HDG with Other Numerical Methods

The accuracy of high-order HDG approximations has been recently exploited to develop efficient algorithms coupling different numerical methodologies in different regions of the computational domain.

In [172], a strategy coupling HDG and a vertex-centred finite volume method is proposed to simulate transient inviscid flows using coarse meshes designed for steady-state problems. In addition, different couplings of HDG and continuous Galerkin (CG) discretisations were explored in the literature. A strategy inspired by a non-overlapping DD method is presented in [208] in the context of incompressible Navier–Stokes flows coupled with conjugate heat transfer phenomena. An alternative minimally-intrusive coupling based on a Nitsche's formulation of the CG method was first introduced in [175] for linear elastic problems involving nearly incompressible materials and was extended to FSI problems with weakly compressible flows in [176].

2.15 HDG-Based Reduced Order Models

In recent years, the accuracy of the HDG method and its flexibility to devise high-order adaptive discretisation have been also employed to devise high-fidelity reduced and surrogate models. In [260, 261], a reduced order model to accelerate the Monte-Carlo simulation of stochastic elliptic PDEs

is constructed coupling a high-order HDG method with a reduced basis and empirical interpolation approach. The combination of an HDG solver for time-harmonic Maxwell's equations and a proper orthogonal decomposition (POD) strategy to design parametrised plasmonic nanogap structures is proposed in [259]. An HGD-POD reduced order model (ROM) is also discussed in [249] for the fast simulation of the unsteady heat equation. More recently, an a priori ROM based on HDG and the proper generalised decomposition was proposed to simulate Stokes flows in geometrically parametrised domains [147, 240].

2.16 Availability of Open-Source Implementations of Hybrid Discretisation Methods

The success of hybrid discretisation methods led to the development of targeted open-source libraries and to their implementation in existing finite element libraries available open-source. To the best of the authors' knowledge, the hybridised DG method based on primal formulations is available in the following libraries:

- `MFEM` [14, 20]
- `Netgen/NGSolve` [11, 235]

whereas the libraries

- `deal.II` [12, 25]
- `Feel++` [4, 213]
- `Firedrake` [6, 219]
- `Nektar++` [10, 43]

provide implementations of the HDG method based on mixed formulations. Finally, the HHO method is available in

- `code_aster` [1, 211]
- `Code_Saturne` [2, 137]
- `DiSk++` [3, 71]
- `GetFEM` [7, 221]
- `HArDCore` [8]

All above mentioned libraries rely on either Fortran or C/C++ implementations, whereas open-source libraries implementing HDG in MATLAB include:

- `HDG3D` [9, 143]
- `FESTUNG` [5, 166]

3 HDG Formulation of the Poisson Equation

In this section, the formulation of the HDG method for the Poisson equation is briefly recalled. Special attention is devoted to the identification of the building blocks of the numerical scheme whose implementation will be detailed in Sect. 8. Interested readers are referred to [89] for a complete theoretical introduction to the HDG method for Poisson equation and to [246] for a tutorial on its derivation.

Let $\Omega \subset \mathbb{R}^{\mathtt{n_{sd}}}$ be an open bounded domain in $\mathtt{n_{sd}}$ spatial dimensions such that its boundary is $\partial\Omega = \Gamma_{_D} \cup \Gamma_{_N}$ and $\Gamma_{_D} \cap \Gamma_{_N} = \emptyset$. The strong form of the Poisson equation is

$$\begin{cases} -\boldsymbol{\nabla}\cdot(\kappa\boldsymbol{\nabla}u) = s & \text{in } \Omega, \\ u = u_{_D} & \text{on } \Gamma_{_D}, \\ \boldsymbol{n}\cdot\kappa\boldsymbol{\nabla}u = g & \text{on } \Gamma_{_N}, \end{cases} \tag{1}$$

where the unknown u represents the solution field, κ denotes the material parameter (e.g. conductivity in a thermal problem) and s is a volumetric source term. On the boundary, Dirichlet, $u_{_D}$, and Neumann, g, data prescribe the values of the unknown and its flux on $\Gamma_{_D}$ and $\Gamma_{_N}$, respectively. The vector $\boldsymbol{n}$ denotes the outward unit normal vector to the boundary.

3.1 HDG Local and Global Problems: Strong Form

Consider a partition of Ω in $\mathtt{n_{el}}$ disjoint subdomains such that

$$\Omega = \bigcup_{e=1}^{\mathtt{n_{el}}} \Omega_e, \quad \Omega_i \cap \Omega_j = \emptyset \text{ for } i \neq j$$

and define the *mesh skeleton* as

$$\Gamma := \left[\bigcup_{e=1}^{\mathtt{n_{el}}} \partial\Omega_e\right] \setminus \partial\Omega.$$

Following the HDG rationale [89, 92, 200, 201, 204, 246], a mixed variable $\boldsymbol{q} = -\sqrt{\kappa}\boldsymbol{\nabla}u$ is introduced and problem (1) is rewritten as a system of first-order equations element-by-element, that is,

$$\begin{cases} \boldsymbol{q} + \sqrt{\kappa}\boldsymbol{\nabla}u = \boldsymbol{0} & \text{in } \Omega_e,\ e = 1,\dots,\mathtt{n_{el}}, \\ \boldsymbol{\nabla}\cdot(\sqrt{\kappa}\boldsymbol{q}) = s & \text{in } \Omega_e,\ e = 1,\dots,\mathtt{n_{el}}, \\ u = u_{_D} & \text{on } \Gamma_{_D}, \\ \boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q} = -g & \text{on } \Gamma_{_N}, \\ [\![u\boldsymbol{n}]\!] = \boldsymbol{0} & \text{on } \Gamma, \\ [\![\boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q}]\!] = 0 & \text{on } \Gamma, \end{cases} \tag{2}$$

where the *jump* operator $[\![\cdot]\!]$ is defined as

$$[\![\odot]\!] := \odot_i + \odot_j,$$

being $\odot_i$ and $\odot_j$ the evaluations of the quantity $\odot$ in two neighbouring elements Ω_i and Ω_j sharing a given interface [191]. The last two conditions in (2), known as *transmission conditions*, enforce the continuity of the solution and of its normal flux across the internal mesh skeleton Γ.

The HDG algorithm solves Eq. (2) in two stages. First, an independent hybrid variable $\hat{u}$ is introduced to represent the trace of the solution on $\partial\Omega_e \backslash \Gamma_{_D}$ and the primal and mixed variables $(u_e, \boldsymbol{q}_e)$ in each element Ω_e, $e = 1, \dots, \mathtt{n_{el}}$ are expressed as functions of the unknown $\hat{u}$, namely

$$\begin{cases} \boldsymbol{q}_e + \sqrt{\kappa}\boldsymbol{\nabla} u_e = \boldsymbol{0} & \text{in } \Omega_e,\ e = 1, \dots, \mathtt{n_{el}}, \\ \boldsymbol{\nabla}\cdot(\sqrt{\kappa}\boldsymbol{q}_e) = s & \text{in } \Omega_e,\ e = 1, \dots, \mathtt{n_{el}}, \\ u_e = u_{_D} & \text{on } \partial\Omega_e \cap \Gamma_{_D}, \\ u_e = \hat{u} & \text{on } \partial\Omega_e \backslash \Gamma_{_D}. \end{cases} \tag{3}$$

Remark 1 Equation (3) represents the $\mathtt{n_{el}}$ HDG local problems. This stage corresponds to the *hybridisation* of the mixed problem, see [138], and is equivalent to the *static condensation* procedure in classical continuous Galerkin methods [157].

Second, the hybrid variable is computed by solving the HDG global problem, which accounts for the transmission conditions on the mesh skeleton Γ and the Neumann boundary condition on $\Gamma_{_N}$, that is,

$$\begin{cases} [\![u\boldsymbol{n}]\!] = \boldsymbol{0} & \text{on } \Gamma, \\ [\![\boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q}]\!] = 0 & \text{on } \Gamma, \\ \boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q} = -g & \text{on } \Gamma_{_N}. \end{cases} \tag{4}$$

Remark 2 The first condition is automatically fulfilled owing to the Dirichlet boundary condition $u_e = \hat{u}$ on $\partial\Omega_e \backslash \Gamma_{_D}$ imposed in the local problem and to the uniqueness of the hybrid variable on each mesh face (respectively, edge in 2D).

The solution $(u_e, \boldsymbol{q}_e)$ in each element Ω_e, $e = 1, \dots, \mathtt{n_{el}}$ is thus efficiently retrieved by solving $\mathtt{n_{el}}$ independent problems, see Eq. (3), element-by-element.

3.2 HDG Local and Global Problems: Weak Form

Following the rationale introduced in [246], the discrete functional spaces

$$\begin{aligned} \mathcal{V}^h(\Omega) := \{ & v \in \mathcal{L}_2(\Omega) : \ v|_{\Omega_e} \in \mathcal{P}^{\mathtt{p}}(\Omega_e) \\ & \forall \Omega_e,\ e = 1, \dots, \mathtt{n_{el}} \}, \end{aligned} \tag{5a}$$

$$\begin{aligned} \widehat{\mathcal{V}}^h(S) := \{ & \hat{v} \in \mathcal{L}_2(S) : \ \hat{v}|_{\Gamma_i} \in \mathcal{P}^{\mathtt{p}}(\Gamma_i) \\ & \forall \Gamma_i \subset S \subseteq \Gamma \cup \partial\Omega \}, \end{aligned} \tag{5b}$$

are defined for the approximation of the element-based and face-based variables, respectively. In (5), $\mathcal{P}^{\mathtt{p}}(\Omega_e)$ and $\mathcal{P}^{\mathtt{p}}(\Gamma_i)$ stand for the spaces of polynomial functions of complete degree at most $\mathtt{p}$ in Ω_e and on Γ_i, respectively.

For $e = 1, \dots, \mathtt{n_{el}}$, the weak form of the HDG local problem is: given $u_{_D}$ on $\Gamma_{_D}$ and $\hat{u}^h$ on $\Gamma \cup \Gamma_{_N}$, find $(u_e^h, \boldsymbol{q}_e^h) \in \mathcal{V}^h(\Omega_e) \times \left[\mathcal{V}^h(\Omega_e)\right]^{\mathtt{n_{sd}}}$ that satisfy

$$\begin{aligned} & -(\boldsymbol{w}, \boldsymbol{q}_e^h)_{\Omega_e} + (\boldsymbol{\nabla}\cdot(\sqrt{\kappa}\boldsymbol{w}), u_e^h)_{\Omega_e} \\ & \quad = \langle \boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{w}, u_{_D} \rangle_{\partial\Omega_e \cap \Gamma_{_D}} + \langle \boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{w}, \hat{u}^h \rangle_{\partial\Omega_e \backslash \Gamma_{_D}}, \end{aligned} \tag{6a}$$

$$\begin{aligned} & (v, \boldsymbol{\nabla}\cdot(\sqrt{\kappa}\boldsymbol{q}_e^h))_{\Omega_e} + \langle v, \tau u_e^h \rangle_{\partial\Omega_e} \\ & \quad = (v, s)_{\Omega_e} + \langle v, \tau u_{_D} \rangle_{\partial\Omega_e \cap \Gamma_{_D}} + \langle v, \tau \hat{u}^h \rangle_{\partial\Omega_e \backslash \Gamma_{_D}}, \end{aligned} \tag{6b}$$

for all $(v, \boldsymbol{w}) \in \mathcal{V}^h(\Omega_e) \times \left[\mathcal{V}^h(\Omega_e)\right]^{\mathtt{n_{sd}}}$, where $(\cdot,\cdot)_D$ and $\langle \cdot, \cdot \rangle_S$ denote the $\mathcal{L}_2$ inner products on a generic subdomain $D \subset \Omega$ and $S \subset \Gamma \cup \partial\Omega$, respectively.

Remark 3 In Eq. (6), τ represents a stabilisation parameter influencing accuracy, stability and convergence of the HDG method [89, 92, 200, 201, 204].

Similarly, the weak form of the HDG global problem is: find $\hat{u}^h \in \widehat{\mathcal{V}}^h(\Gamma \cup \Gamma_{_N})$ that satisfies

$$\begin{aligned} \sum_{e=1}^{\mathtt{n_{el}}} \Big\{ & \langle \hat{v}, \boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q}_e^h \rangle_{\partial\Omega_e \backslash \Gamma_{_D}} + \langle \hat{v}, \tau u_e^h \rangle_{\partial\Omega_e \backslash \Gamma_{_D}} \\ & - \langle \hat{v}, \tau \hat{u}^h \rangle_{\partial\Omega_e \backslash \Gamma_{_D}} \Big\} = - \sum_{e=1}^{\mathtt{n_{el}}} \langle \hat{v}, g \rangle_{\partial\Omega_e \cap \Gamma_{_N}}, \end{aligned} \tag{7}$$

for all $\hat{v} \in \widehat{\mathcal{V}}^h(\Gamma \cup \Gamma_{_N})$.

3.3 HDG Local and Global Problems: Discrete Form

An isoparametric formulation is considered for the primal, mixed and hybrid variables in the discrete spaces (5), that is,

$$u^h = \sum_{i=1}^{\mathtt{n_{en}}} N_i \mathtt{u}_i, \quad \boldsymbol{q}^h = \sum_{i=1}^{\mathtt{n_{en}}} N_i \mathbf{q}_i, \quad \hat{u}^h = \sum_{i=1}^{\mathtt{n_{fn}}} \hat{N}_i \hat{\mathtt{u}}_i, \tag{8}$$

where $\mathtt{u}_i$, $\mathbf{q}_i$ and $\hat{\mathtt{u}}_i$ are the nodal values of the unknowns, N_i and $\hat{N}_i$ are the polynomial shape functions of degree $\mathtt{p}$ defined in a reference element and on a reference face, respectively and $\mathtt{n_{en}}$ and $\mathtt{n_{fn}}$ denote the number of nodes per element and per face, respectively.

Hence, for each element Ω_e, $e = 1, \dots, \mathtt{n_{el}}$, the local problem (6) leads to the linear system of equations

$$\begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_e \begin{Bmatrix} \mathbf{u} \\ \mathbf{q} \end{Bmatrix}_e = \begin{Bmatrix} \mathbf{f}_u \\ \mathbf{f}_q \end{Bmatrix}_e + \begin{bmatrix} \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{q\hat{u}} \end{bmatrix}_e \hat{\mathbf{u}}_e, \tag{9}$$

from which the following solution is computed

$$\begin{Bmatrix} \mathbf{u} \\ \mathbf{q} \end{Bmatrix}_e = \begin{Bmatrix} \mathbf{z}_u^f \\ \mathbf{z}_q^f \end{Bmatrix}_e + \begin{bmatrix} \mathbf{Z}_{u\hat{u}} \\ \mathbf{Z}_{q\hat{u}} \end{bmatrix}_e \hat{\mathbf{u}}_e, \tag{10a}$$

with the matrices

$$\begin{bmatrix} \mathbf{Z}_{u\hat{u}} \\ \mathbf{Z}_{q\hat{u}} \end{bmatrix}_e := \begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_e^{-1} \begin{bmatrix} \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{q\hat{u}} \end{bmatrix}_e \tag{10b}$$

and the vectors

$$\begin{Bmatrix} \mathbf{z}_u^f \\ \mathbf{z}_q^f \end{Bmatrix}_e := \begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_e^{-1} \begin{Bmatrix} \mathbf{f}_u \\ \mathbf{f}_q \end{Bmatrix}_e. \tag{10c}$$

The corresponding discretisation of the global problem (7) leads to

$$\sum_{e=1}^{\mathtt{n_{el}}} \left\{ \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_e \begin{Bmatrix} \mathbf{u} \\ \mathbf{q} \end{Bmatrix}_e + [\mathbf{A}_{\hat{u}\hat{u}}]_e \hat{\mathbf{u}}_e \right\} = \sum_{e=1}^{\mathtt{n_{el}}} [\mathbf{f}_{\hat{u}}]_e. \tag{11}$$

By plugging the local elemental solution (10a) into (11), the HDG problem

$$\mathbf{K}\hat{\mathbf{u}} = \hat{\mathbf{f}}$$

involving only the globally-coupled degrees of freedom is obtained, where the matrix and the right-hand side vector are obtained by assembling the elemental contributions

$$\hat{\mathbf{K}}^e := \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_e \begin{bmatrix} \mathbf{Z}_{u\hat{u}} \\ \mathbf{Z}_{q\hat{u}} \end{bmatrix}_e + [\mathbf{A}_{\hat{u}\hat{u}}]_e, \tag{12a}$$

$$\hat{\mathbf{f}}^e := [\mathbf{f}_{\hat{u}}]_e - \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_e \begin{Bmatrix} \mathbf{z}_u^f \\ \mathbf{z}_q^f \end{Bmatrix}_e. \tag{12b}$$

The expressions of the matrices and vectors introduced in this section are detailed in Appendix A.

3.4 HDG Local Postprocess

Introduce the discrete functional space

$$\begin{aligned} \mathcal{V}_\star^h(\Omega) := \{ & v \in \mathcal{L}_2(\Omega) : \ v|_{\Omega_e} \in \mathcal{P}^{\mathtt{p}+1}(\Omega_e) \\ & \forall \Omega_e,\ e = 1, \dots, \mathtt{n_{el}} \}, \end{aligned} \tag{13}$$

where $\mathcal{P}^{\mathtt{p}+1}(\Omega_e)$ denotes the space of polynomial functions of complete degree at most $\mathtt{p}+1$ in each element Ω_e.

The HDG postprocess procedure allows to compute a superconvergent approximation $u_\star$ of the primal variable by solving an independent local problem in each element, namely

$$\begin{cases} -\boldsymbol{\nabla}\cdot(\kappa\boldsymbol{\nabla} u_\star) = \boldsymbol{\nabla}\cdot(\sqrt{\kappa}\boldsymbol{q}_e) & \text{in } \ \Omega_e, \\ \boldsymbol{n}\cdot\kappa\boldsymbol{\nabla} u_\star = -\boldsymbol{n}\cdot\sqrt{\kappa}\boldsymbol{q}_e & \text{on } \ \partial\Omega_e, \end{cases} \tag{14}$$

with the constraint

$$(u_\star, 1)_{\Omega_e} = (u_e, 1)_{\Omega_e} \tag{15}$$

on the mean value of the solution in the element.

For each element Ω_e, $e = 1, \dots, \mathtt{n_{el}}$, the weak form of the postprocess procedure is: find $u_\star^h \in \mathcal{V}_\star^h(\Omega_e)$ that satisfies

$$\left(\boldsymbol{\nabla} v_\star, \kappa\boldsymbol{\nabla} u_\star^h\right)_{\Omega_e} = -\left(\boldsymbol{\nabla} v_\star, \sqrt{\kappa}\boldsymbol{q}_e^h\right)_{\Omega_e}, \tag{16a}$$

$$(u_\star^h, 1)_{\Omega_e} = (u_e^h, 1)_{\Omega_e}, \tag{16b}$$

for all $v_\star \in \mathcal{V}_\star^h(\Omega_e)$.

Using an isoparametric approximation for the functions in the space $\mathcal{V}_\star^h(\Omega)$, the HDG local postprocess gives rise to the linear system

$$\begin{bmatrix} \mathbf{A}_{\star\star} & \mathbf{a}_{\star\lambda} \\ \mathbf{a}_{\star\lambda}^T & 0 \end{bmatrix}_e \begin{Bmatrix} \mathbf{u}_\star \\ \lambda \end{Bmatrix}_e = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{\star q} \\ \mathbf{a}_{\star\lambda}^T & \mathbf{0} \end{bmatrix}_e \begin{Bmatrix} \mathcal{I}^\star \mathbf{u} \\ \mathcal{I}_{\mathtt{n_{sd}}}^\star \mathbf{q} \end{Bmatrix}_e, \tag{17}$$

where the saddle-point structure of the problem follows from the imposition of the constraint (16b) via the Lagrange multiplier λ and $\mathcal{I}^\star : \mathcal{V}^h \to \mathcal{V}_\star^h$ and $\mathcal{I}_{\mathtt{n_{sd}}}^\star : [\mathcal{V}^h]^{\mathtt{n_{sd}}} \to [\mathcal{V}_\star^h]^{\mathtt{n_{sd}}}$ denote the interpolation operators from the spaces of polynomial functions of degree $\mathtt{p}$ to the ones of degree $\mathtt{p}+1$ for scalar and vector-valued functions.

The expressions of the matrices and vectors introduced in this section are detailed in Appendix A.

4 HDG Formulation of the Stokes Equations

This section presents the formulation of the HDG method for the Stokes equations, extending the framework previously introduced for the Poisson equation. For the sake of simplicity, the present work focuses on the velocity-pressure formulation of the Stokes equations. For the Cauchy stress tensor formulation, a tutorial to devise an HDG method based on equal order approximation for all the variables and pointwise symmetric mixed variable is presented in [151].

The open bounded domain $\Omega \subset \mathbb{R}^{\mathtt{n_{sd}}}$ is characterised now by a boundary partitioned in three portions disjoint by pairs such that $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_S$, where Dirichlet, Neumann and slip conditions are imposed. The strong form of the Stokes equations is

$$\begin{cases} -\boldsymbol{\nabla}\cdot(\nu\boldsymbol{\nabla}\boldsymbol{u} - p\mathbf{I}_{\mathtt{n_{sd}}}) = \boldsymbol{s} & \text{in } \Omega, \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{u}_{_D} & \text{on } \Gamma_{_D}, \\ \boldsymbol{n}\cdot(\nu\boldsymbol{\nabla}\boldsymbol{u} - p\mathbf{I}_{\mathtt{n_{sd}}}) = \boldsymbol{g} & \text{on } \Gamma_{_N}, \\ \boldsymbol{u}\cdot\boldsymbol{D} + \boldsymbol{n}\cdot\big(\nu\boldsymbol{\nabla}\boldsymbol{u} - p\mathbf{I}_{\mathtt{n_{sd}}}\big)\boldsymbol{E} = \mathbf{0} & \text{on } \Gamma_{_S}, \end{cases} \tag{18}$$

where the pair $(\boldsymbol{u}, p)$ denotes the unknown velocity and pressure fields, $\nu > 0$ is the kinematic viscosity, $\boldsymbol{n}$ is the outward unit normal to the boundary, $\boldsymbol{s}$ is the vector of the body forces and $\boldsymbol{u}_{_D}$ and $\boldsymbol{g}$ represent the imposed velocity and pseudo-traction on the Dirichlet and Neumann boundaries, respectively. On the slip boundary, matrices $\boldsymbol{D}$ and $\boldsymbol{E}$ are defined as $\boldsymbol{D} := [\boldsymbol{n}, \beta\boldsymbol{t}_1, \dots, \beta\boldsymbol{t}_{\mathtt{n_{sd}}-1}]$ and $\boldsymbol{E} := [\alpha\boldsymbol{n}, \boldsymbol{t}_1, \dots, \boldsymbol{t}_{\mathtt{n_{sd}}-1}]$, the tangential vectors $\boldsymbol{t}_j$, $j = 1, \dots, \mathtt{n_{sd}} - 1$ being such that $\{\boldsymbol{n}, \boldsymbol{t}_1, \dots, \boldsymbol{t}_{\mathtt{n_{sd}}-1}\}$ form an orthonormal system of vectors. Two scalars, α and β, represent the penetration and friction coefficient, respectively. For $\alpha, \beta \to 0$, the case of a perfectly slip condition is retrieved [151].

Remark 4 The divergence-free condition in Eq. (18) induces the following compatibility condition on the velocity field

$$\langle \boldsymbol{u}_{_D}\cdot\boldsymbol{n}, 1\rangle_{\Gamma_{_D}} + \langle \boldsymbol{u}\cdot\boldsymbol{n}, 1\rangle_{\partial\Omega\setminus\Gamma_{_D}} = 0. \tag{19}$$

Remark 5 In case of a purely Dirichlet boundary value problem, that is $\partial\Omega = \Gamma_{_D}$, an additional constraint needs to be introduced to retrieve uniqueness of the pressure field. A common approach relies on imposing the mean value of the pressure in the domain [120, 218]

$$\frac{1}{|\Omega|}(p, 1)_{\Omega} = 0, \tag{20}$$

or on the boundary of the domain [88, 98, 148, 199], namely

$$\frac{1}{|\partial\Omega|}\langle p, 1\rangle_{\partial\Omega} = 0. \tag{21}$$

4.1 HDG Local and Global Problems: Strong Form

Following the rationale presented in Sect. 3, Eq. (18) is rewritten element-by-element as a system of first-order equations by introducing a mixed variable $\boldsymbol{L} = -\sqrt{\nu}\boldsymbol{\nabla}\boldsymbol{u}$, namely

$$\begin{cases} \boldsymbol{L} + \sqrt{\nu}\boldsymbol{\nabla}\boldsymbol{u} = \mathbf{0} & \text{in } \Omega_e, \\ \boldsymbol{\nabla}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big) = \boldsymbol{s} & \text{in } \Omega_e, \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 & \text{in } \Omega_e, \\ \boldsymbol{u} = \boldsymbol{u}_{_D} & \text{on } \Gamma_{_D}, \\ \boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big) = -\boldsymbol{g} & \text{on } \Gamma_{_N}, \\ \boldsymbol{u}\cdot\boldsymbol{D} - \boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big)\boldsymbol{E} = \mathbf{0} & \text{on } \Gamma_{_S}, \\ [\![\boldsymbol{u}\otimes\boldsymbol{n}]\!] = \mathbf{0} & \text{on } \Gamma, \\ [\![\boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big)]\!] = \mathbf{0} & \text{on } \Gamma, \end{cases} \tag{22}$$

for $e = 1, \dots, \mathtt{n_{el}}$.

First, the HDG algorithm performes the hybridisation step by expressing $(\boldsymbol{u}_e, p_e, \boldsymbol{L}_e)$ in each element Ω_e as functions of the unknown trace of the velocity $\widehat{\boldsymbol{u}}$ on the element faces via the HDG local problem

$$\begin{cases} \boldsymbol{L}_e + \sqrt{\nu}\boldsymbol{\nabla}\boldsymbol{u}_e = \mathbf{0} & \text{in } \Omega_e, \\ \boldsymbol{\nabla}\cdot\big(\sqrt{\nu}\boldsymbol{L}_e + p_e\mathbf{I}_{\mathtt{n_{sd}}}\big) = \boldsymbol{s} & \text{in } \Omega_e, \\ \boldsymbol{\nabla}\cdot\boldsymbol{u}_e = 0 & \text{in } \Omega_e, \\ \boldsymbol{u}_e = \boldsymbol{u}_{_D} & \text{on } \partial\Omega_e\cap\Gamma_{_D}, \\ \boldsymbol{u}_e = \widehat{\boldsymbol{u}} & \text{on } \partial\Omega_e\setminus\Gamma_{_D}, \end{cases} \tag{23a}$$

for $e = 1, \dots, \mathtt{n_{el}}$. Note that Eq. (23a) is a purely Dirichlet boundary value problem. Hence, following Remark 5, the constraint

$$\frac{1}{|\partial\Omega_e|}\langle p_e, 1\rangle_{\partial\Omega} = \rho_e, \quad \text{for } e = 1, \dots, \mathtt{n_{el}}, \tag{23b}$$

is introduced, where ρ_e is an independent variable representing the mean value of the pressure on the boundary $\partial\Omega_e$. It is worth noting that the variable ρ was not present in the HDG approximation of the Poisson equation and its treatment in the `HDGlab` code will be detailed in Sect. 9.

The HDG global problem thus accounts for the transmission and non-Dirichlet boundary conditions, that is

$$\begin{cases} [\![\boldsymbol{u}\otimes\boldsymbol{n}]\!] = \mathbf{0} & \text{on } \Gamma, \\ [\![\boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big)]\!] = \mathbf{0} & \text{on } \Gamma, \\ \boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big) = -\boldsymbol{g} & \text{on } \Gamma_{_N}, \\ \boldsymbol{u}\cdot\boldsymbol{D} - \boldsymbol{n}\cdot\big(\sqrt{\nu}\boldsymbol{L} + p\mathbf{I}_{\mathtt{n_{sd}}}\big)\boldsymbol{E} = \mathbf{0} & \text{on } \Gamma_{_S}, \end{cases} \tag{24a}$$

where, following Remark 2, the first condition is automatically fulfilled. In addition, the constraint in Remark 4 is rewritten element-by-element in terms of the hybrid variable $\widehat{\boldsymbol{u}}$ leading to

$$\langle \boldsymbol{u}_{_D}\cdot\boldsymbol{n}, 1\rangle_{\partial\Omega_e\cap\Gamma_{_D}} + \langle \widehat{\boldsymbol{u}}\cdot\boldsymbol{n}, 1\rangle_{\partial\Omega_e\setminus\Gamma_{_D}} = 0, \tag{24b}$$

for $e = 1, \dots, \mathtt{n_{el}}$.

4.2 HDG Local and Global Problems: Weak Form

The corresponding weak form of the HDG local problem (23) is: given $\boldsymbol{u}_{_D}$ on $\Gamma_{_D}$ and $\widehat{\boldsymbol{u}}^h$ on $\Gamma \cup \Gamma_{_N} \cup \Gamma_{_S}$, find $(\boldsymbol{L}_e^h, \boldsymbol{u}_e^h, p_e^h) \in [\mathcal{V}^h(\Omega_e)]^{\mathtt{n_{sd}}\times\mathtt{n_{sd}}} \times [\mathcal{V}^h(\Omega_e)]^{\mathtt{n_{sd}}} \times \mathcal{V}^h(\Omega_e)$ that satisfy

$$
\begin{cases}
-(\boldsymbol{W}, \boldsymbol{L}_e^h)_{\Omega_e} + (\boldsymbol{\nabla}\cdot(\sqrt{\nu}\boldsymbol{W}), \boldsymbol{u}_e^h)_{\Omega_e} \\
\quad = \langle \boldsymbol{n}_e\cdot\sqrt{\nu}\boldsymbol{W}, \boldsymbol{u}_{_D}\rangle_{\partial\Omega_e\cap\Gamma_{_D}} + \langle \boldsymbol{n}_e\cdot\sqrt{\nu}\boldsymbol{W}, \widehat{\boldsymbol{u}}^h\rangle_{\partial\Omega_e\setminus\Gamma_{_D}}, \\
(\boldsymbol{w}, \boldsymbol{\nabla}\cdot(\sqrt{\nu}\boldsymbol{L}_e^h))_{\Omega_e} + (\boldsymbol{w}, \boldsymbol{\nabla} p_e^h)_{\Omega_e} + \langle \boldsymbol{w}, \tau\boldsymbol{u}_e^h\rangle_{\partial\Omega_e} \\
\quad = (\boldsymbol{w}, \boldsymbol{s})_{\Omega_e} + \langle \boldsymbol{w}, \tau\boldsymbol{u}_{_D}\rangle_{\partial\Omega_e\cap\Gamma_{_D}} + \langle \boldsymbol{w}, \tau\widehat{\boldsymbol{u}}^h\rangle_{\partial\Omega_e\setminus\Gamma_{_D}}, \\
(\boldsymbol{\nabla} q, \boldsymbol{u}_e^h)_{\Omega_e} = \langle q, \boldsymbol{u}_{_D}\cdot\boldsymbol{n}_e\rangle_{\partial\Omega_e\cap\Gamma_{_D}} + \langle q, \widehat{\boldsymbol{u}}^h\cdot\boldsymbol{n}_e\rangle_{\partial\Omega_e\setminus\Gamma_{_D}}, \\
\langle p_e^h, 1\rangle_{\partial\Omega_e} = |\partial\Omega_e|\rho_e^h,
\end{cases}
\tag{25}
$$

for all $(\boldsymbol{W}, \boldsymbol{v}, q) \in [\mathcal{V}^h(\Omega_e)]^{\mathtt{n_{sd}}\times\mathtt{n_{sd}}} \times [\mathcal{V}^h(\Omega_e)]^{\mathtt{n_{sd}}} \times \mathcal{V}^h(\Omega_e)$. It is worth noting that, differently from the Poisson case, Eq. (25) provides $(\boldsymbol{L}_e^h, \boldsymbol{u}_e^h, p_e^h)$ in terms of two global unknowns, $\widehat{\boldsymbol{u}}^h$ and $\rho^h := \{\rho_1^h, \ldots, \rho_{\mathtt{n_{el}}}^h\}^T$.

Similarly, the weak form of the HDG global problem (24) is: find $\widehat{\boldsymbol{u}}^h \in [\widehat{\mathcal{V}}^h]^{\mathtt{n_{sd}}}$ and $\boldsymbol{\rho}^h \in \mathbb{R}^{\mathtt{n_{el}}}$ such that

$$
\begin{cases}
\displaystyle\sum_{e=1}^{\mathtt{n_{el}}} \Big\{ \langle \widehat{\boldsymbol{v}}, \boldsymbol{n}_e\cdot\sqrt{\nu}\boldsymbol{L}_e^h\rangle_{\partial\Omega_e\setminus(\Gamma_{_D}\cup\Gamma_{_S})} \\
\qquad - \langle \widehat{\boldsymbol{v}}, \boldsymbol{n}_e\cdot\sqrt{\nu}\boldsymbol{L}_e^h\boldsymbol{E}\rangle_{\partial\Omega_e\cap\Gamma_{_S}} \\
\qquad + \langle \widehat{\boldsymbol{v}}, p_e^h\boldsymbol{n}_e\rangle_{\partial\Omega_e\setminus(\Gamma_{_D}\cup\Gamma_{_S})} \\
\qquad - \langle \widehat{\boldsymbol{v}}, p_e^h\boldsymbol{n}_e\cdot\boldsymbol{E}\rangle_{\partial\Omega_e\cap\Gamma_{_S}} \\
\qquad + \langle \widehat{\boldsymbol{v}}, \tau\boldsymbol{u}_e^h\rangle_{\partial\Omega_e\setminus(\Gamma_{_D}\cup\Gamma_{_S})} \\
\qquad - \langle \widehat{\boldsymbol{v}}, \tau\boldsymbol{u}_e^h\cdot\boldsymbol{E}\rangle_{\partial\Omega_e\cap\Gamma_{_S}} \\
\qquad - \langle \widehat{\boldsymbol{v}}, \tau\widehat{\boldsymbol{u}}^h\rangle_{\partial\Omega_e\setminus(\Gamma_{_D}\cup\Gamma_{_S})} \\
\qquad + \langle \widehat{\boldsymbol{v}}, \widehat{\boldsymbol{u}}^h\cdot(\boldsymbol{D}+\tau\boldsymbol{E})\rangle_{\partial\Omega_e\cap\Gamma_{_S}} \Big\} \\
\quad = -\displaystyle\sum_{e=1}^{\mathtt{n_{el}}} \langle \widehat{\boldsymbol{v}}, \boldsymbol{g}\rangle_{\partial\Omega_e\cap\Gamma_{_N}}, \\
\langle \widehat{\boldsymbol{u}}^h\cdot\boldsymbol{n}_e, 1\rangle_{\partial\Omega_e\setminus\Gamma_{_D}} = -\langle \boldsymbol{u}_{_D}\cdot\boldsymbol{n}_e, 1\rangle_{\partial\Omega_e\cap\Gamma_{_D}}, \\
\qquad \text{for} \quad e = 1, \ldots, \mathtt{n_{el}}
\end{cases}
\tag{26}
$$

for all $\widehat{\boldsymbol{v}} \in [\widehat{\mathcal{V}}^h]^{\mathtt{n_{sd}}}$.

4.3 HDG Local and Global Problems: Discrete Form

The discretisation of the local problem (25) leads to the following linear system of equations

$$
\begin{bmatrix}
\mathbf{A}_{LL} & \mathbf{A}_{Lu} & \mathbf{0} & \mathbf{0} \\
\mathbf{A}_{Lu}^T & \mathbf{A}_{uu} & \mathbf{A}_{pu}^T & \mathbf{0} \\
\mathbf{0} & \mathbf{A}_{pu} & \mathbf{0} & \mathbf{a}_{\rho p} \\
\mathbf{0} & \mathbf{0} & \mathbf{a}_{\rho p}^T & 0
\end{bmatrix}_e
\begin{Bmatrix}
\mathbf{L} \\ \mathbf{u} \\ \mathbf{p} \\ \zeta
\end{Bmatrix}_e
= \begin{Bmatrix}
\mathbf{f}_L \\ \mathbf{f}_u \\ \mathbf{f}_p \\ 0
\end{Bmatrix}_e
+ \begin{bmatrix}
\mathbf{A}_{L\hat{u}} \\ \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{p\hat{u}} \\ \mathbf{0}
\end{bmatrix}_e
\widehat{\mathbf{u}}_e
+ \begin{Bmatrix}
\mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ 1
\end{Bmatrix}_e
\rho_e,
\tag{27}
$$

where the constraint (23b) is imposed using the Lagrange multiplier ζ. It is worth noting that the Lagrange multiplier is required to guarantee that Eq. (27) is well-posed and the computed pressure field is unique but is not utilised in the solution of the global HDG problem. The resulting local elemental solution is given by

$$
\begin{Bmatrix}
\mathbf{L} \\ \mathbf{u} \\ \mathbf{p} \\ \zeta
\end{Bmatrix}_e
= \begin{Bmatrix}
\mathbf{z}_L^f \\ \mathbf{z}_u^f \\ \mathbf{z}_p^f \\ \mathbf{z}_\zeta^f
\end{Bmatrix}_e
+ \begin{bmatrix}
\mathbf{Z}_{L\hat{u}} \\ \mathbf{Z}_{u\hat{u}} \\ \mathbf{Z}_{p\hat{u}} \\ \mathbf{Z}_{\zeta\hat{u}}
\end{bmatrix}_e
\widehat{\mathbf{u}}_e
+ \begin{Bmatrix}
\mathbf{z}_L^\rho \\ \mathbf{z}_u^\rho \\ \mathbf{z}_p^\rho \\ \mathbf{z}_\zeta^\rho
\end{Bmatrix}_e
\rho_e,
\tag{28a}
$$

with the matrix and vectors defined as

$$
\begin{bmatrix}
\mathbf{Z}_{L\hat{u}} \\ \mathbf{Z}_{u\hat{u}} \\ \mathbf{Z}_{p\hat{u}} \\ \mathbf{Z}_{\zeta\hat{u}}
\end{bmatrix}_e
:= \begin{bmatrix}
\mathbf{A}_{LL} & \mathbf{A}_{Lu} & \mathbf{0} & \mathbf{0} \\
\mathbf{A}_{Lu}^T & \mathbf{A}_{uu} & \mathbf{A}_{pu}^T & \mathbf{0} \\
\mathbf{0} & \mathbf{A}_{pu} & \mathbf{0} & \mathbf{a}_{\rho p} \\
\mathbf{0} & \mathbf{0} & \mathbf{a}_{\rho p}^T & 0
\end{bmatrix}_e^{-1}
\begin{bmatrix}
\mathbf{A}_{L\hat{u}} \\ \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{p\hat{u}} \\ \mathbf{0}
\end{bmatrix}_e,
\tag{28b}
$$

$$
\begin{Bmatrix}
\mathbf{z}_L^f \\ \mathbf{z}_u^f \\ \mathbf{z}_p^f \\ \mathbf{z}_\zeta^f
\end{Bmatrix}_e
:= \begin{bmatrix}
\mathbf{A}_{LL} & \mathbf{A}_{Lu} & \mathbf{0} & \mathbf{0} \\
\mathbf{A}_{Lu}^T & \mathbf{A}_{uu} & \mathbf{A}_{pu}^T & \mathbf{0} \\
\mathbf{0} & \mathbf{A}_{pu} & \mathbf{0} & \mathbf{a}_{\rho p} \\
\mathbf{0} & \mathbf{0} & \mathbf{a}_{\rho p}^T & 0
\end{bmatrix}_e^{-1}
\begin{Bmatrix}
\mathbf{f}_L \\ \mathbf{f}_u \\ \mathbf{f}_p \\ 0
\end{Bmatrix}_e,
\tag{28c}
$$

$$
\begin{Bmatrix}
\mathbf{z}_L^\rho \\ \mathbf{z}_u^\rho \\ \mathbf{z}_p^\rho \\ \mathbf{z}_\zeta^\rho
\end{Bmatrix}_e
:= \begin{bmatrix}
\mathbf{A}_{LL} & \mathbf{A}_{Lu} & \mathbf{0} & \mathbf{0} \\
\mathbf{A}_{Lu}^T & \mathbf{A}_{uu} & \mathbf{A}_{pu}^T & \mathbf{0} \\
\mathbf{0} & \mathbf{A}_{pu} & \mathbf{0} & \mathbf{a}_{\rho p} \\
\mathbf{0} & \mathbf{0} & \mathbf{a}_{\rho p}^T & 0
\end{bmatrix}_e^{-1}
\begin{Bmatrix}
\mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ 1
\end{Bmatrix}_e.
\tag{28d}
$$

The discrete form of the global problem (26) reads as

$$\begin{aligned}&\sum_{e=1}^{\mathtt{n_{el}}}\left\{\left[\begin{array}{ccc}\mathbf{A}_{\hat{u}L} & \mathbf{A}_{\hat{u}u} & \mathbf{A}_{\hat{u}p}\end{array}\right]_e\left\{\begin{array}{c}\mathbf{L}\\ \mathbf{u}\\ \mathbf{p}\end{array}\right\}_e\right.\\&\left.+\,[\mathbf{A}_{\hat{u}\hat{u}}]_e\,\widehat{\mathbf{u}}_e\right\}=\sum_{e=1}^{\mathtt{n_{el}}}[\mathbf{f}_{\hat{u}}]_e,\\&\mathbf{1}^T\,[\mathbf{A}_{p\hat{u}}]_e\widehat{\mathbf{u}}_e=-\mathbf{1}^T\,[\mathbf{f}_p]_e.\end{aligned}\tag{29}$$

By inserting the solution (28a) into (29), the following system involving the globally-coupled unknowns $\widehat{\mathbf{u}}$ and $\boldsymbol{\rho}$ is obtained

$$\left[\begin{array}{cc}\widehat{\mathbf{K}} & \mathbf{G}\\ \mathbf{G}^T & \mathbf{0}\end{array}\right]\left\{\begin{array}{c}\widehat{\mathbf{u}}\\ \boldsymbol{\rho}\end{array}\right\}=\left\{\begin{array}{c}\widehat{\mathbf{f}}_{\hat{u}}\\ \widehat{\mathbf{f}}_{\rho}\end{array}\right\},\tag{30}$$

where the matrices and vectors are obtained by assembling the elemental contributions

$$\widehat{\mathbf{K}}^e:=\left[\begin{array}{cccc}\mathbf{A}_{\hat{u}L} & \mathbf{A}_{\hat{u}u} & \mathbf{A}_{\hat{u}p} & \mathbf{0}\end{array}\right]_e\left[\begin{array}{c}\mathbf{Z}_{L\hat{u}}\\ \mathbf{Z}_{u\hat{u}}\\ \mathbf{Z}_{p\hat{u}}\\ \mathbf{Z}_{\zeta\hat{u}}\end{array}\right]_e+[\mathbf{A}_{\hat{u}\hat{u}}]_e,\tag{31a}$$

$$\mathbf{G}^T:=\left[\begin{array}{c}\mathbf{1}^T\,[\mathbf{A}_{p\hat{u}}]_1\\ \mathbf{1}^T\,[\mathbf{A}_{p\hat{u}}]_2\\ \cdots\\ \mathbf{1}^T\,[\mathbf{A}_{p\hat{u}}]_{\mathtt{n_{el}}}\end{array}\right],\tag{31b}$$

$$\widehat{\mathbf{f}}^e_{\hat{u}}:=[\mathbf{f}_{\hat{u}}]_e-\left[\begin{array}{cccc}\mathbf{A}_{\hat{u}L} & \mathbf{A}_{\hat{u}u} & \mathbf{A}_{\hat{u}p} & \mathbf{0}\end{array}\right]_e\left\{\begin{array}{c}\mathbf{z}^f_L\\ \mathbf{z}^f_u\\ \mathbf{z}^f_p\\ \mathbf{z}^f_\zeta\end{array}\right\}_e,\tag{31c}$$

$$\widehat{\mathbf{f}}_{\rho}:=-\left[\begin{array}{c}\mathbf{1}^T\,[\mathbf{f}_p]_1\\ \mathbf{1}^T\,[\mathbf{f}_p]_2\\ \cdots\\ \mathbf{1}^T\,[\mathbf{f}_p]_{\mathtt{n_{el}}}\end{array}\right].\tag{31d}$$

The expressions of the matrices and vectors introduced above are detailed in Appendix B.

Remark 6 Differently from the Poisson case, the global problem (30) features a saddle-point structure, as classical in the context of incompressible flows [120]. The proof of the symmetry of the HDG matrix in Eq. (30) can be devised following the rationale described in [151].

4.4 HDG Local Postprocess

The HDG postprocess procedure presented in Sect. 3.4 for the Poisson equation can be extended straightforwardly to the case of the vectorial variable $\boldsymbol{u}$.

Remark 7 The postprocessing procedure utilised here is inspired by the work of Stenberg [253] and was exploited in the framework of HDG to obtain an improved approximation of the velocity field via the solution of an additional inexpensive element-by-element problem [199, 247]. Nonetheless, in the context of incompressible flows, it is often of interest retrieving an $H(\mathrm{div})$-conforming and exactly divergence-free approximation of the velocity field. For this purpose, alternative postprocessing strategies inspired by the Brezzi–Douglas–Marini (BDM) projection operator [36] were proposed [90, 92]. It is worth noting that the above mentioned procedures are suitable only for the velocity-pressure formulation of the incompressible flow equations. In case a formulation based on the Cauchy stress tensor is considered, an additional constraint is required to handle the rigid rotational modes as discussed in [148, 151, 245].

A superconvergent velocity field $\boldsymbol{u}_\star$ is thus obtained by solving an independent local problem in each element, namely

$$\left\{\begin{array}{ll}-\boldsymbol{\nabla}\cdot(\nu\boldsymbol{\nabla}\boldsymbol{u}_\star)=\boldsymbol{\nabla}\cdot(\sqrt{\nu}\boldsymbol{L}_e) & \text{in }\ \Omega_e,\\ \boldsymbol{n}\cdot\nu\boldsymbol{\nabla}\boldsymbol{u}_\star=-\boldsymbol{n}\cdot\sqrt{\nu}\boldsymbol{L}_e & \text{on }\ \partial\Omega_e,\end{array}\right.\tag{32}$$

with the constraint

$$(\boldsymbol{u}_\star,1)_{\Omega_e}=(\boldsymbol{u}_e,1)_{\Omega_e}\tag{33}$$

on the mean value of the velocity in the element.

Hence, for each element Ω_e, $e=1,\dots,\mathtt{n_{el}}$, the weak form of the postprocess procedure is: find $\boldsymbol{u}^h_\star\in[\mathcal{V}^h_\star(\Omega_e)]^{\mathtt{n_{sd}}}$ such that

$$(\boldsymbol{\nabla}\boldsymbol{v}_\star,\nu\boldsymbol{\nabla}\boldsymbol{u}^h_\star)_{\Omega_e}=-(\boldsymbol{\nabla}\boldsymbol{v}_\star,\sqrt{\nu}\boldsymbol{L}^h_e)_{\Omega_e},\tag{34a}$$

$$(\boldsymbol{u}^h_\star,1)_{\Omega_e}=(\boldsymbol{u}^h_e,1)_{\Omega_e},\tag{34b}$$

for all $\boldsymbol{v}_\star\in[\mathcal{V}^h_\star(\Omega_e)]^{\mathtt{n_{sd}}}$.

Using an isoparametric approximation for the functions in the space $[\mathcal{V}^h_\star(\Omega)]^{\mathtt{n_{sd}}}$, the HDG local postprocess for the Stokes equations leads to

$$\left[\begin{array}{cc}\mathbf{A}_{\star\star} & \mathbf{A}_{\star\lambda}\\ \mathbf{A}^T_{\star\lambda} & \mathbf{0}\end{array}\right]_e\left\{\begin{array}{c}\mathbf{u}_\star\\ \lambda\end{array}\right\}_e=\left[\begin{array}{cc}\mathbf{0} & \mathbf{A}_{\star L}\\ \mathbf{A}^T_{\star\lambda} & \mathbf{0}\end{array}\right]_e\left\{\begin{array}{c}\mathcal{I}^\star_{\mathtt{n_{sd}}}\mathbf{u}\\ \mathcal{I}^\star_{\mathtt{n_{sd}}\times\mathtt{n_{sd}}}\mathbf{L}\end{array}\right\}_e,\tag{35}$$

where λ is the vector of Lagrange multipliers imposing the constraint (34b) and $\mathcal{I}^\star_{\mathtt{n_{sd}}\times\mathtt{n_{sd}}}:[\mathcal{V}^h]^{\mathtt{n_{sd}}\times\mathtt{n_{sd}}}\to[\mathcal{V}^h_\star]^{\mathtt{n_{sd}}\times\mathtt{n_{sd}}}$ denotes

the interpolation operator for tensor-valued functions from the space of polynomials of degree p to the one of degree p + 1.

The expressions of the matrices and vectors introduced above are detailed in Appendix B.

5 The HDGlab Repository

The implementation of the HDG solver for the Poisson and Stokes equations has been made available as an open-source software, released under the terms of the GNU General Public License version 3.0 or any later version (https://www.gnu.org/licenses) and is freely available from the repository: https://git.lacan.upc.edu/hybridLab/HDGlab.

The structure of the repository, illustrated in Fig. 1, is described in this section.

The directory dat contains the data files. This includes two and three dimensional meshes in the directories meshes2D and meshes3D respectively, the pre-computed reference elements in two and three dimensions in the directory refElem and the data structure required to postprocess high-order HDG solutions in the directory postprocess.

The directory Poisson contains the HDG solver for the Poisson problem as described in Sect. 3. The directory hdgPoisson contains the core HDG library for the Poisson equation. The directory testsPoisson is used to organise the functions that describe the setup of the problems to be solved, including the definition of the boundary conditions, source term and, if known, the analytical solution. The directory resPoisson is where the results are saved after an execution of the Poisson solver.

The directory Stokes contains the HDG solver for the Stokes problem as described in Sect. 4. The structure of this directory follows the same rationale as the one corresponding to the Poisson solver.

The directory common contains a set of functions that are common to both the Poisson and the Stokes solvers.

Finally, the directory importMesh contains a library that is provided to import a mesh generated with the open source software Gmsh [146] in HDGlab. The core routines to convert a mesh from *.msh* to *.mat* format are located in the directory GMSH. The directory examples contains some test cases including *.geo* and *.msh* files, whereas the output of the imported mesh is stored in the directory meshFiles. This library is described in detail in Appendix C.

6 Data Structures

Three data structures are used to manage the mesh, the reference element and the face information required to compute the elemental contributions of the global HDG problem.

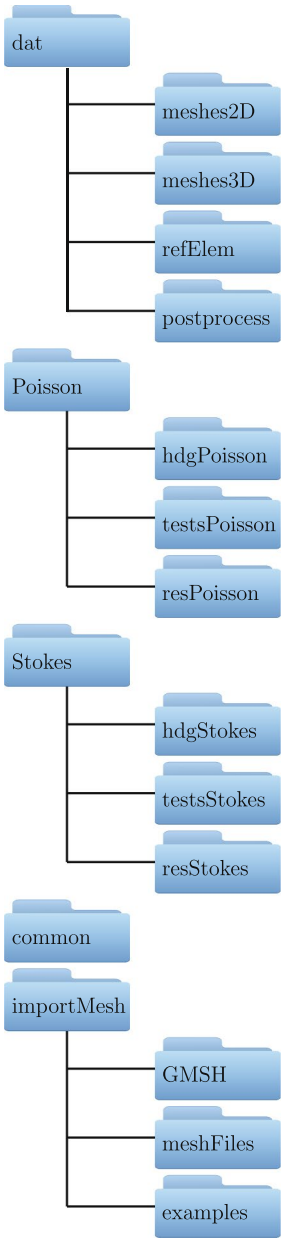


Fig. 1 Structure of the HDGlab repository

These three variables are assumed to be an input of the HDG library and a detailed description is provided in this section. To make the developed software more accessible variables of type *struct* are used in this work rather than *class* types.

6.1 Mesh

The variable mesh contains the following information:

- nsd: Number of spatial dimensions.
- optionNodes: Type of high-order nodal distribution, being an equally-spaced (0) or a Fekete (1) nodal set.
- nOfNodes: Number of nodes.

- `X`: Array of dimension `nOfNodes`×`nsd` containing the nodal coordinates of the mesh.
- `nOfElements`: Number of elements.
- `indexT`: Array of dimension `nOfElements`×2 containing the connectivity indices of the mesh. The first column contains the first node of the element and the second column contains the last node of the element.
- `pElem`: Array of dimension 1×`nOfElements` containing the degree of approximation of each element.
- `matElem`: Array of dimension 1×`nOfElements` containing the material flag for each element.
- `nOfIntFaces`: Number of interior faces (i.e. faces not on the boundary).
- `intFaces`: Array of dimension `nOfIntFaces`×5. The first two columns contain the first element sharing this face and its local face number. The next two columns contain the second element sharing this face and its local face number. The last column contains the local node number of the second face that matches with the first local node of the first face.
- `nOfExtFaces`: Number of exterior faces (i.e. faces on the boundary).
- `extFaces`: Array of dimension `nOfIntFaces`×4. The first two columns contains the element sharing this face and its local face number. The third column contains the boundary condition flag and the last column contains the flag of the boundary curve or surface.

The information stored in the field `intFaces` is characteristic of a DG formulation, where integrals on interior faces need to be computed. This is in contrast with a standard CG formulation, where only the field `extFaces` is needed to impose the boundary conditions. The last column of `intFaces` is needed to account for the different orientation of an interior face as seen from the element on the left and on the right of a face. It is worth noting that in two dimensions the information could be omitted because the local node number of the second face that matches with the first local node of the first face is always equal to two, as illustrated in Fig. 2. However, in three dimensions this information is required as there are three possible rotations of the local face nodes that do not alter the orientation of the element/face, as depicted in Fig. 3.

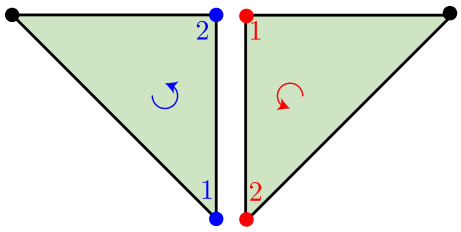


Fig. 2 Interface between two triangular elements, showing the orientation of each element and the local node number of each node on the face as seen from the element on the left and on the right of the interface

To illustrate the `mesh` data structure, a coarse mesh of the domain $\Omega = [0,1]^2$, with four triangular elements is considered. Figure 4 represents the mesh with the global numbering of the mesh elements, the element nodes, the mesh faces and the face nodes.

An overview of the data contained in the `mesh` data structure for the example of Fig. 4 is shown in Fig. 5, with the details of some of the arrays depicted in Fig. 6.

It is worth noting that the connectivity of an element is obtained by using the function shown in Fig. 7. This means that the mesh nodes in the array `X` are duplicated. This functionality is meant to help the handling of meshes with a non-uniform degree of approximation and to provide a seamless

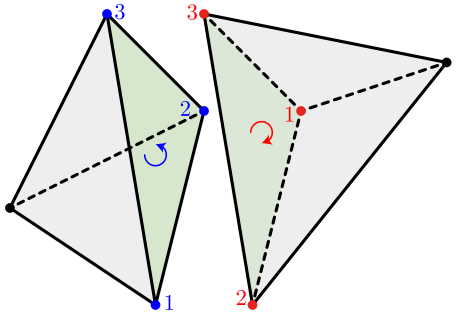


Fig. 3 Interface between two tetrahedral elements, showing the orientation of each face and the local node number of each node on the face as seen from the element on the left and on the right of the interface

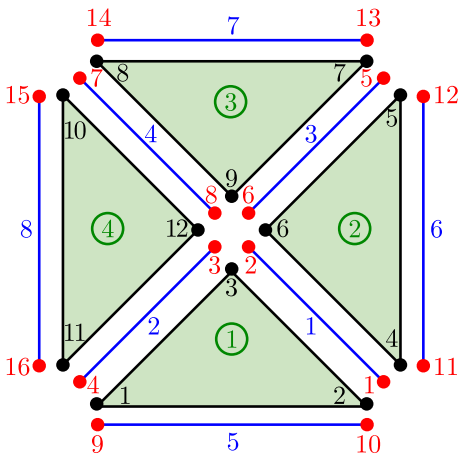


Fig. 4 Mesh of $\Omega = [0,1]^2$ with four triangular elements with the global numbering of the mesh elements (in green), the element nodes (in black), the mesh faces (in blue) and the face nodes (in red). (Color figure online)

```
>> disp(mesh)

nsd:            2
optionNodes:    0
nOfNodes:       12
X:              [12x2 double]
nOfElements:     4
indexT:         [4x2 double]
pElem:          [1 1 1 1]
matElem:        [1 1 1 1]
nOfIntFaces:    4
intFaces:       [4x5 double]
nOfExtFaces:    4
extFaces:       [4x4 double]
```

Fig. 5 Overview of the data contained in the `mesh` data structure for the example of Fig. 4

```
>> disp(mesh.X)
0.0    0.0
1.0    0.0
0.5    0.5
1.0    0.0
1.0    1.0
0.5    0.5
1.0    1.0
0.0    1.0
0.5    0.5
0.0    1.0
0.0    0.0
0.5    0.5

>> disp(mesh.indexT)
1     3
4     6
7     9
10    12

>> disp(mesh.intFaces)
1     2     2     3     2
1     3     4     2     2
2     2     3     3     2
3     2     4     3     2

>> disp(mesh.extFaces)
1     1     1     1
2     1     1     2
3     1     1     3
4     1     1     4
```

Fig. 6 Detail of the fields `X`, `indexT`, `intFaces` and `extFaces`, corresponding to the data structure `mesh` of Fig. 5

```
function Te = getElemConnectivity(mesh, iElem)

Te = mesh.indexT(iElem,1):mesh.indexT(iElem,2);
```

Fig. 7 Function to retrieve the connectivity of an element

way to partition the mesh in case future users would like to parallelise the code. If desired, a different array can be introduced for the connectivity information and the user only needs to redefine the function `getElemConnectivity`.

6.2 Reference Element

As usual in an isoparametric finite element context, the information related to the approximation and numerical integration in an element is stored by means of a reference element, with local coordinates, $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_{\mathtt{n_{sd}}})$. To easily handle meshes with a non-uniform degree of approximation, the variable `refElem` is considered an array of dimension $1 \times \mathtt{p_{max}}$, where $\mathtt{p_{max}}$ is the maximum degree of approximation used in all the elements. For each component of the `refElem` array, the following information is stored:

- `nsd`: Number of spatial dimensions.
- `optionNodes`: Type of high-order nodal distribution, being an equally-spaced (0) or a Fekete (1) nodal set.
- `p`: Degree of approximation.
- `nOfVertices`: Number of element vertices.
- `nOfNodes`: Number of element nodes.
- `coordinates`: Array of dimension `nOfNodes` $\times$ `nsd` containing the local nodal coordinates.
- `nOfFaces`: Number of element faces.
- `face`: Array of dimension $1 \times$ `nOfFaces`. Each position is a structure containing the following information about an element face:
  - `nodes`: Array containing the element local number of the nodes in the current face.
  - `nodesPerm`: Array containing `nsd` $+ 1$ permutations of the face nodes in the field `nodes`, using the element local number. Each row provides the permutation as required by the field `intFaces` in the `mesh` data structure.
  - `nodesPermHDG`: Array containing `nsd` $+ 1$ permutations of the face nodes in the field `nodes`, using the face local number. Each row provides the permutation as required by the field `intFaces` in the `mesh` data structure.
- `nOfGauss`: Number of integration points.
- `gaussWeights`: Array of dimension `nOfGauss` $\times 1$, containing the integration weights.
- `shapeFunctions`: Array of dimension `nOfGauss` $\times$ `nOfNodes` $\times$ (`nsd` $+ 1$). When the third index is equal to 1, the array contains the value of the shape functions, for all the nodes at all integration points. When the third index is equal to $r > 1$, the array contains the value of the derivative of the shape functions in the ξ_{r-1} direction, for all the nodes at all integration points.
- `shapeFunctionsNodesPPp1`: Array of dimension `nOfNodes`$^\star \times$ `nOfNodes`, where `nOfNodes`* denotes the number of nodes of an element with degree of approximation $\mathtt{p} + 1$. It contains the value of the shape functions of the current element at the nodes of an element with one extra degree of approximation. This array is only used to perform the HDG postprocess described in Sects. 3.4 and 4.4.
- `shapeFunctionsGaussPPp1`: Array of dimension `nOfGauss`$^\star \times$ `nOfNodes`, where `nOfGauss`* denotes the number of integration points of an element with

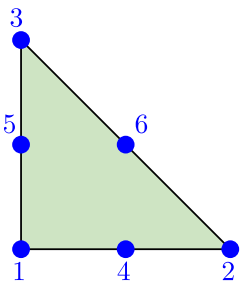


Fig. 8 Reference triangular element for p = 2

```
>> disp(refElem(2))
nsd:                    2
optionNodes:            0
p:                      2
nOfVertices:            3
nOfNodes:               6
coordinates:            [6x2 double]
nOfFaces:               3
face:                   [1x3 struct]
nOfGauss:               7
gaussWeights:           [7x1 double]
shapeFunctions:         [7x6x3 double]
shapeFunctionsNodesPPp1: [10x6 double]
shapeFunctionsGaussPPp1: [15x6x3 double]
```

Fig. 9 Overview of the data contained in the `refElem` data structure for the reference triangular quadratic element of Fig. 8

degree of approximation p + 1. It contains the value of the shape functions of the current element and its derivatives at the integration points of an element with one extra degree of approximation. This array is only used to perform the HDG postprocess described in Sects. 3.4 and 4.4.

To illustrate the `refElem` data structure, Fig. 8 depicts the reference quadratic triangular element. An overview of the data contained in the `refElem` data structure for the example of Fig. 8 is shown in Fig. 9.

Similarly, Fig. 10 shows the reference quadratic tetrahedral element and Fig. 11 depicts and overview of the data contained in the `refElem` data structure.

It is worth noting that the code provided utilises nodal basis functions for the polynomial approximation. However, it is straightforward for future users to change the basis used for the approximation by simply changing the reference element information. With minimum effort it is also possible to incorporate other element types.

6.3 Reference Face

The information related to the approximation and numerical integration on a face is stored by means of a reference

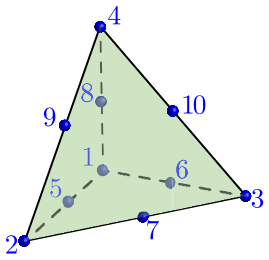


Fig. 10 Reference tetrahedral element for p = 2

```
>> disp(refElem(2))
nsd:                    3
optionNodes:            0
p:                      2
nOfVertices:            4
nOfNodes:               10
coordinates:            [10x3 double]
nOfFaces:               4
face:                   [1x4 struct]
nOfGauss:               14
gaussWeights:           [14x1 double]
shapeFunctions:         [14x10x4 double]
shapeFunctionsNodesPPp1: [20x10 double]
shapeFunctionsGaussPPp1: [35x10x4 double]
```

Fig. 11 Overview of the data contained in the `refElem` data structure for the reference tetrahedral quadratic element of Fig. 10

```
>> disp(refFace(2,2))
nsd:            1
optionNodes:    0
p:              2
nOfVertices:    2
nOfNodes:       3
coordinates:    [-1  0  1]
nOfGauss:       3
gaussPoints:    [3x1 double]
gaussWeights:   [3x1 double]
shapeFunctions: [3x3x3 double]
```

Fig. 12 Overview of the data contained in the `refFace` data structure for the reference face of a quadratic element in two dimensions

face, with local coordinates, $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_{\mathtt{n_{sd}}-1})$. To easily handle meshes with a non-uniform degree of approximation, the variable `refFace` is considered an array of dimension $\mathtt{p_{max}} \times \mathtt{p_{max}}$, where $\mathtt{p_{max}}$ is the maximum degree of approximation used in all the elements. For each diagonal component of the `refFace` array, the information stored is a subset of the information stored in the `refElem` data structure. As an example, Fig. 12 offers an overview of the data contained in the diagonal term of `refFace` corresponding to a quadratic face of a triangular element.

The upper triangular portion of the `refFace` data structure contains the information associated with the field `shapeFunctions`. This is only required when a mesh with a non-uniform degree of approximation is employed. The position (r, s) of the array `refFace` contains the

shape functions of degree r evaluated at the integration points of a face with degree of approximation s. This is required when computing the integrals in an interior face where the degree of approximation of the elements sharing this face is different. It is worth noting that only the entries in the upper triangle of the array `refFace` contain relevant information because the degree of approximation used for the hybrid variable in a given face is defined as the maximum between the degree of approximation of the two elements sharing the face.

7 Preprocess

This section describes the preprocessing stage of the HDG solver. The implementation can be found in the function `hdgPreprocess`, which produces as an output an updated version of the data structures `mesh` and `hdg`.

The data structure `mesh` is taken as an input, containing the fields described in Sect. 6.1, and a new field, called `indexTf` is added. This field contains an array of dimension `nOfFaces` $\times 2$ featuring the connectivity indices of the mesh skeleton, where `nOfFaces` is the total number of mesh faces (i.e. interior faces plus exterior faces). The first column contains the first degree of freedom of a face and the second column contains the last degree of freedom of the face. Figure 13 shows the data contained in `indexTf`, after the preprocessing is performed, for the mesh of Fig. 4 and for a Poisson problem (i.e. scalar unknown). The same information for a Stokes problem is shown in Fig. 14.

For the Poisson problem, each node has one degree of freedom associated as the hybrid variable contains an approximation of the trace of the solution, which is a scalar quantity. In contrast, for the Stokes problem the global vector of unknowns contains an approximation of the trace of the velocity and an approximation of the mean pressure. Therefore, in two dimensions each face contains $2(p+1)$ degrees of freedom for the velocity and one extra degree of freedom per element is introduced for the mean pressure.

```
>> disp(mesh.indexTf)
     1     2
     3     4
     5     6
     7     8
     9    10
    11    12
    13    14
    15    16
```

Fig. 13 Data contained in `mesh.indexTf` data structure for the solution of the Poisson equation in the mesh of Fig. 4

```
>> disp(mesh.indexTf)
     1     4
     5     8
     9    12
    13    16
    17    20
    21    24
    25    28
    29    32
```

Fig. 14 Data contained in `mesh.indexTf` data structure for the solution of the Stokes equation in the mesh of Fig. 4

The `hdg` data structure is also an input of the function `hdgPreprocess`, containing two user defined parameters, namely:

- `tau`: Value of the constant stabilisation parameter.
- `problem`: String containing the name of the problem to be solved. The code provided contains two model problems, namely `'Poisson'` and `'Stokes'`.

The structure is updated in the preprocess stage with the following information:

- `faceInfo`: Structure of dimension 1 × `nOfElements`. For each element of the array, the following information provides a link between the element and face information of the mesh:
  - `local2global`: Array of dimension 1 × `nOfElementFaces`, containing the global face number of the faces of the current element.
  - `localNumFlux`: Array of dimension 1 × `nOfElementFaces`, containing a flag for the numerical flux function associated with the faces of the current element. For an interior face, a value of zero is set. For a boundary face, the number corresponds to the boundary condition to be imposed and it is linked to the third column of the array `extFaces` of the `mesh` data structure described in Sect. 6.1.
  - `permutations`: Array of dimension 1 × `nOfElementFaces`, containing a flag for the permutation required to ensure that the ordering of the nodes in the global face matches the ordering of the face nodes in the current element.
  - `pHat`: Array of dimension 1 × `nOfElementFaces`, containing the degree of approximation used for the hybrid variable in the corresponding faces of the current element.
- `nDOFglobal`: Number of global degrees of freedom.

```
>> disp(hdg)
tau:            1
problem:        'Poisson'
faceInfo:       [1x4 struct]
nDOFglobal:     16
vDOFtoSolve:    [1 2 3 4 5 6 7 8]
```

Fig. 15 Data contained in the `hdg` data structure for the mesh of Fig. 4

```
>> disp(hdg.faceInfo(1))
local2global:   [5 1 2]
localNumFlux:   [1 0 0]
permutations:   [0 0 0]
pHat:           [1 1 1]

>> disp(hdg.faceInfo(2))
local2global:   [6 3 1]
localNumFlux:   [1 0 0]
permutations:   [0 0 2]
pHat:           [1 1 1]
```

Fig. 16 Data contained in the `hdg.faceInfo` for the first two elements of the mesh of Fig. 4

- `vDOFtoSolve`: Vector of global degrees of freedom associated with nodes not on a Dirichlet boundary.

In the case of the Stokes equations, three additional fields are introduced in the `hdg` data structure during the preprocess routine:

- `pureDirichlet`: Boolean variable identifying whether the problem has purely Dirichlet boundary conditions.
- `columnsGlobalSystem`: Number of columns in the global system, corresponding to the number of unknowns given by the hybrid velocity and the mean pressure.
- `rowsGlobalSystem`: Number of rows in the global system, including the constraint for the uniqueness of pressure. The value of this variable will differ from `columnsGlobalSystem` only in the case of purely Dirichlet boundary value problems.

The data contained in the `hdg` data structure, after the preprocess stage, is shown in Fig. 15 for the Poisson problem on the mesh of Fig. 4. The data contained in the field `faceInfo` for the first two elements is also depicted in Fig. 16.

Two more simple data structures are defined at the preprocess stage, namely `problemParams` and `ctt`. The structure `problemParams` contains problem-specific information. The current implementation uses this data structure to carry the following information:

- `nOfMat`: Number of materials in the domain.
- `charLength`: A characteristic length, used to define the value of the stabilisation parameter of the HDG formulation.
- `example`: An integer that points to the number of a user-defined example.

In addition, `problemParams` stores the information on the material parameters. For the Poisson problem:

- `conductivity`: Array of dimension $1 \times$ `nOfMat` that contains the conductivity of each material present in the domain.

For the Stokes problem:

- `viscosity`: A scalar value representing the viscosity coefficient of the fluid.
- `alphaSlip`: A scalar value describing the penetration coefficient for the slip boundary condition.
- `betaSlip`: A scalar value describing the friction coefficient for the slip boundary condition.

Finally, the structure `ctt` contains the following information:

- `iBC_Interior`: A flag for the numerical flux function to be used on an interior face.
- `iBC_Dirichlet`: A flag for the numerical flux function to be used on an exterior face where a Dirichlet boundary condition is imposed.
- `iBC_Neumann`: A flag for the numerical flux function to be used on an exterior face where a Neumann boundary condition is imposed.
- `iBC_Slip`: A flag for the numerical flux function to be used on an exterior face where a slip boundary condition is imposed (only supported for the Stokes problem).
- `nOfComponents`: Number of components of the primal variable.

The flags used to distinguish the type of face and numerical flux to be considered can be specified by the user and they are linked to the third column of the array `extFaces` of the `mesh` data structure described in Sect. 6.1.

8 The `HDGlab` Poisson Solver

A code workflow diagram of the `HDGlab` Poisson code is shown in Fig. 17. This section focuses on the core part of the code that involves the assembly and solution of the global system of equations, the element-by-element solution of the

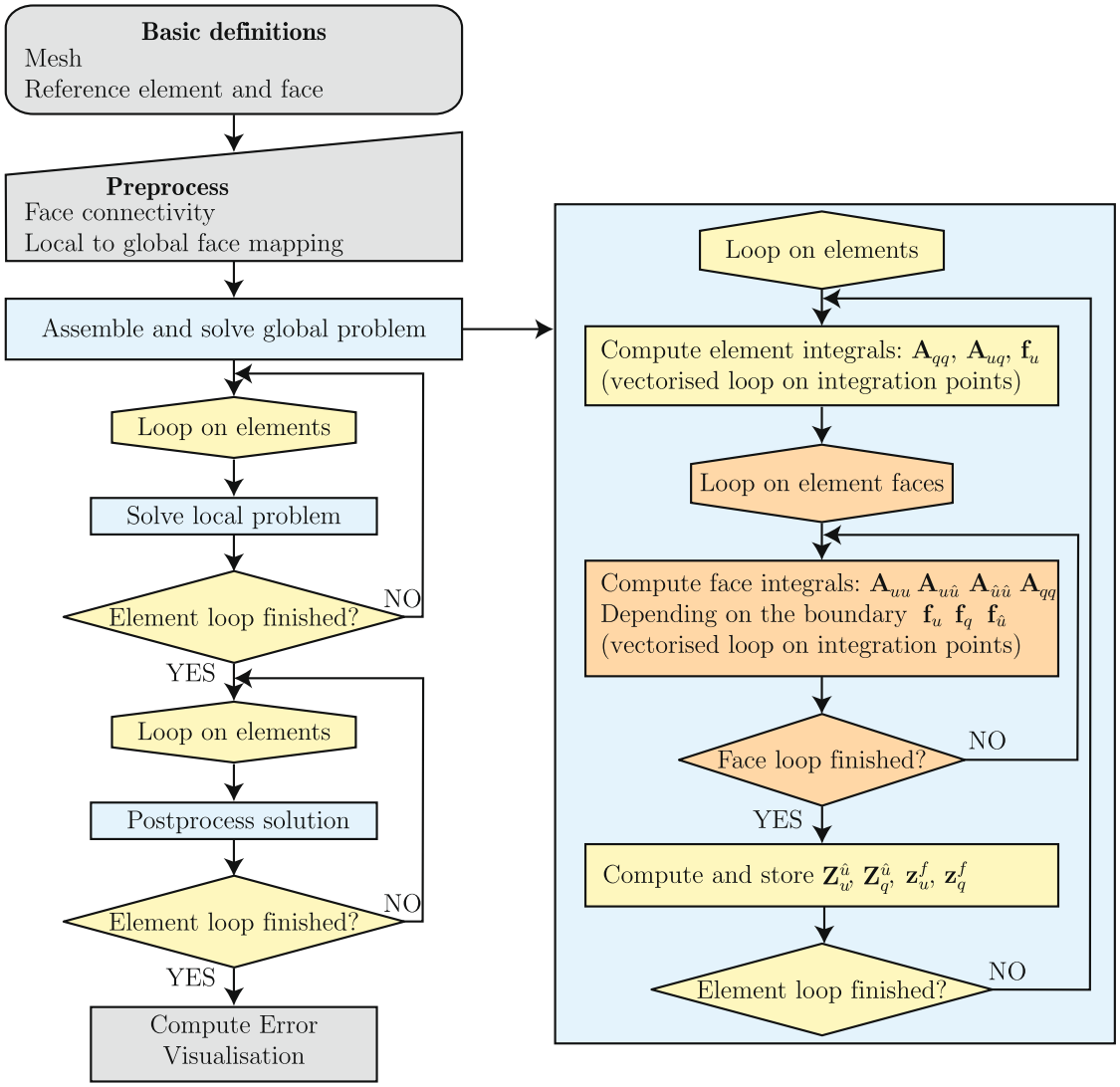


Fig. 17 Code workflow diagram

local problems and the local postprocess to obtain a superconvergent solution.

8.1 Global Problem

The HDG global system of equations is assembled and solved in the function `hdg_Poisson_GlobalSystem`. For each element, `hdg_Poisson_ElementalMatrices` contains two parts corresponding to the computation of the element integrals and the face integrals respectively. An extract of this function, showing the computation of the elemental matrices $\mathbf{A}_{qq}$ and $\mathbf{A}_{uq}$ and the elemental vector $\mathbf{f}_u$, is displayed in Fig. 18. It is worth noting that the loop on integration points is vectorised by using the binary singleton expansion function `bsxfun`.

In a similar fashion, the second part of the function `hdg_Poisson_ElementalMatrices` performs a loop on the faces of the current element and computes the face integrals that lead to the matrices $\mathbf{A}_{uu}$, $\mathbf{A}_{u\hat{u}}$, $\mathbf{A}_{q\hat{u}}$ and $\mathbf{A}_{\hat{u}\hat{u}}$. This computation distinguishes between interior and exterior faces and, for the exterior faces, utilises the flag in `hdg.faceInfo.localNumFlux` to enforce the correct boundary condition. For a Dirichlet face, the vector $\mathbf{f}_u$ is updated and the vector $\mathbf{f}_q$ is computed. For a Neumann face, the vector $\mathbf{f}_{\hat{u}}$ is computed.

One of the distinctive parts of the HDG formulation is found in the loop on faces, where a vector called `indexFlip` is used to ensure that the ordering of the degrees of freedom corresponding to the hybrid variable, as seen from the current element, matches the global ordering of the

```
% Element computation -----------------------------------------------------
[N, dNx, wXY, Xg] = gaussElemCartesianInfo(refElem(pElem), Xe);
Nw = bsxfun(@times, N, wXY');
minusNN = -Nw*N';

for iNsd = 1:nsd
    vElem = iNsd:nsd:ndofQ;
    Auq(:,vElem)     = sqrt(kappa)*Nw*dNx(:,:,iNsd)';
    Aqq(vElem,vElem) = minusNN;
end

sourceTerm = poisson_source(Xg,problemParams,matElem,nsd,problemParams.example);
fu = Nw*sourceTerm;
```

Fig. 18 Extract of the function `hdg_Poisson_ElementalMatrices` that computes the element integrals of the HDG formulation for the Poisson problem

```
% Elemental mapping -------------------------
Z = [Auu Auq; Auq' Aqq]\[Aul fu; Aql fq];
vU = 1:ndofU;
vQ = ndofU+1:ndofU+ndofQ;
Zul = Z(vU,1:ndofUHat);
zuf = Z(vU,ndofUHat+1);
Zql = Z(vQ,1:ndofUHat);
zqf = Z(vQ,ndofUHat+1);

% Flipping due to the different numbering of
% internal face nodes when seen from left or
% right element
Zql = Zql(:,indexFlip);
Zul = Zul(:,indexFlip);
Alq = Aql(:,indexFlip)';
Alu = Aul(:,indexFlip)';
All = All(indexFlip,indexFlip);
fl  = fl(indexFlip);

% Elemental matrices
Ke = Alq*Zql + Alu*Zul + All;
fe = fl - (Alq*zqf + Alu*zuf);
```

Fig. 19 Extract of the `hdg_Poisson_ElementalMatrices` function that computes the elemental matrix $\widehat{\mathbf{K}}^e$ and vector $\widehat{\mathbf{f}}^e$

degrees of freedom of the vector of unknowns of the global HDG system.

After all the elemental matrices and vectors are computed, the elemental contributions to the global system are prepared to be assembled, namely $\widehat{\mathbf{K}}^e$ and $\widehat{\mathbf{f}}^e$, as shown in the extract of Fig. 19. It is worth noting that at this stage the matrices $\mathbf{Z}_{u\hat{u}}$ and $\mathbf{Z}_{q\hat{u}}$ and the vectors $\mathbf{z}_u^f$ and $\mathbf{z}_q^f$, defined in Eq. (10), are stored in the data structure `local` in order to be used during the solution of the element-by-element local problems. For large problems, the user might choose to save these matrices to disk before solving the global system of equations.

8.2 Local Problem

After the global system of equations is solved, the function `hdg_Poisson_LocalProblem`, shown in Fig. 20, is called to solve the local problems. This function is just the implementation of Eq. (10a). The only aspect that requires attention is the indexing of the global vectors for the primal and mixed variables. This is managed by the function `hdgElemToFaceIndex`, which is designed to work for variables with any number of components. It is also worth noting that this function accounts for the possibility to have a non-uniform degree of approximation.

8.3 Local Postprocess

As discussed in Sect. 3.4, once the primal and mixed variables are computed, it is possible to perform a local, element-by-element, postprocess procedure to obtain a more accurate, superconvergent, approximation of the solution. This is implemented in the function `hdg_Poisson_LocalPostprocess`, shown in Fig. 21.

A key aspect in `hdg_Poisson_LocalPostprocess` is the computation of the elemental matrices and vectors in Eq. (17), which is implemented in the function `hdg_Poisson_LocalPostprocessElemMat`.

It is worth emphasising that the implementation assumes that no extra geometric information is known at this stage. Therefore, to compute the high-order nodal distribution of degree $p+1$, the nodal distribution in the reference element is mapped to the physical space by using the isoparametric mapping of degree p. This implies that, for curved elements, a subparametric formulation is considered. This formulation can lead to a suboptimal rate of convergence for the post-processed solution as demonstrated in [239, 247], where a NURBS-enhanced implementation was proposed.

9 The `HDGlab` Stokes Solver

In this section, the `HDGlab` solver for the Stokes equations is presented. It is worth noting that the code features the same structure introduced in the previous section for the

```
function [u,q] = ...
    hdg_Poisson_LocalProblem(mesh,refElem, ...
    refFace,hdg,uHat,local,nOfComponents)

% Count the number of unknowns
nOfDOFU = 0;
for iElem = 1:mesh.nOfElements
    pElem = mesh.pElem(iElem);
    nOfElementNodes = refElem(pElem).nOfNodes;
    nOfDOFU = nOfDOFU + ...
        nOfElementNodes*nOfComponents;
end

% Initialisation
u = zeros(nOfDOFU,1);
q = zeros(nOfDOFU*mesh.nsd,1);

indexUIni = 1;
indexQIni = 1;
for iElem = 1:mesh.nOfElements
    pElem = mesh.pElem(iElem);

    iGlobalFace = ...
        hdgElemToFaceIndex(mesh.indexTf, ...
        refFace,hdg.faceInfo(iElem), ...
        refElem(pElem).nOfFaces, ...
        nOfComponents);

    nOfElementNodes = refElem(pElem).nOfNodes;
    nDOFsElemU = nOfElementNodes*nOfComponents;
    nDOFsElemQ = nDOFsElemU*mesh.nsd;

    indexUEnd = indexUIni + nDOFsElemU - 1;
    indexQEnd = indexQIni + nDOFsElemQ - 1;

    u(indexUIni:indexUEnd) = ...
        local(iElem).Zul*uHat(iGlobalFace) ...
        + local(iElem).zuf;
    q(indexQIni:indexQEnd) = ...
        local(iElem).Zql*uHat(iGlobalFace) ...
        + local(iElem).zqf;

    indexUIni = indexUEnd + 1;
    indexQIni = indexQEnd + 1;
end
```

Fig. 20 Function `hdg_Poisson_LocalProblem` to solve the element-by-element local problems

Poisson case. Hence, only the differences with respect to the Poisson solver will be detailed.

9.1 A Vector-Valued Problem

The HDG global system of equations is assembled and solved in the function `hdg_Stokes_GlobalSystem`. More precisely, the element and face integrals are computed by the function `hdg_Stokes_ElementalMatrices` for each element.

The first difference with respect to the Poisson code is represented by the vectorial nature of the primal and hybrid variables and by the tensor-valued mixed variable. For the sake of computational efficiency, the representation of the second-order tensor $\mathbf{L}$ is written as a vector by rows, namely

$$\mathbf{L} = \begin{cases} [\mathrm{L}_{11}\ \mathrm{L}_{12}\ \mathrm{L}_{21}\ \mathrm{L}_{22}]^T, & \text{in 2D,} \\ [\mathrm{L}_{11}\ \mathrm{L}_{12}\ \mathrm{L}_{13}\ \mathrm{L}_{21}\ \mathrm{L}_{22}\ \mathrm{L}_{23}\ \mathrm{L}_{31}\ \mathrm{L}_{32}\ \mathrm{L}_{33}]^T, & \text{in 3D.} \end{cases}$$

Figure 22 reports the computation of the elemental matrices $\mathbf{A}_{LL}$, $\mathbf{A}_{Lu}$ and $\mathbf{A}_{pu}$ and the elemental vector $\mathbf{f}_u$. It is worth noting that the assembly of these matrices and this vector accounts for the appropriate numbering of the vector-valued and tensor-valued unknowns. Similarly to the Poisson case, the loop on integration points is vectorised via the command `bsxfun`.

9.2 Slip Boundary Conditions

In the second part of `hdg_Stokes_ElementalMatrices`, the face integrals are computed within a loop on the element faces. More precisely, the matrices $\mathbf{A}_{L\hat{u}}$, $\mathbf{A}_{uu}$, $\mathbf{A}_{u\hat{u}}$ and $\mathbf{A}_{p\hat{u}}$ are computed for the local problem, whereas the matrix $\mathbf{A}_{\hat{u}\hat{u}}$ is computed for the global problem. In addition, on the Neumann faces the vector $\mathbf{f}_{\hat{u}}$ is computed, whereas on the Dirichlet faces the vectors $\mathbf{f}_L$ and $\mathbf{f}_p$ are computed and the vector $\mathbf{f}_u$ is updated.

Remark 8 In case of Dirichlet and Neumann boundary conditions, the remaining matrices involved in the global problem are such that

$$\mathbf{A}_{\hat{u}L} = \mathbf{A}_{L\hat{u}}^T, \tag{36a}$$

$$\mathbf{A}_{\hat{u}u} = \mathbf{A}_{u\hat{u}}^T, \tag{36b}$$

$$\mathbf{A}_{\hat{u}p} = \mathbf{A}_{p\hat{u}}^T. \tag{36c}$$

In case slip boundary conditions are also considered, the properties in Eq. (36) no longer hold and the matrices $\mathbf{A}_{\hat{u}L}$, $\mathbf{A}_{\hat{u}u}$ and $\mathbf{A}_{\hat{u}p}$ are computed in the function `hdg_Stokes_ElementalMatrices`. Figure 23 displays the initialisation of the above elemental matrices which are then computed in the loop on faces when the flag in `hdg.faceInfo.localNumFlux` matches `ctt.iBC_Slip`.

A specific treatment of the case in which slip boundary conditions are considered is also required for the definition of the elemental matrices of the global problem with appropriate ordering of the degrees of freedom of the hybrid variable using the `indexFlip` vector, see Fig. 24.

9.3 Additional Constraint in the Local Problem

A major difference between the Poisson and Stokes case is the structure of the local problems (9) and (27). Despite both matrices are symmetric, the one arising from the discretisation of the Poisson equation is positive definite, whereas it is indefinite in the Stokes case. More precisely, the matrix

```
function uStar = hdg_Poisson_LocalPostprocess(mesh,refElem,u,q,problemParams,nOfComponents)

% Initialisation
nOfDOFUStar = 0;
for iElem = 1:mesh.nOfElements
    pElem = mesh.pElem(iElem);
    pElemStar = pElem + 1;
    nOfElementNodesStar = refElem(pElemStar).nOfNodes;
    nOfDOFUStar = nOfDOFUStar + nOfElementNodesStar*nOfComponents;
end
uStar = zeros(nOfDOFUStar,1);

% Computation
indexIni = 1;
for iElem = 1:mesh.nOfElements
    Te = getElemConnectivity(mesh, iElem);
    Xe = mesh.X(Te,:);
    pElem = mesh.pElem(iElem);
    pElemStar = pElem + 1;

    NXeStar = refElem(pElem).shapeFunctionsNodesPPp1;
    XeStar = NXeStar*Xe;

    iMat = mesh.matElem(iElem);
    kappa = problemParams.conductivity(iMat);

    [ug,qg] = hdg_Poisson_LocalPostprocessInterpolation(NXeStar,u,q,refElem(pElemStar). ...
        nOfNodes,Te,mesh.nsd);
    [Ke,Bqe,intUStar,intU] = ...
        hdg_Poisson_LocalPostprocessElemMat(refElem(pElemStar),XeStar,ug,qg,kappa);

    % Constraint with Lagrange multipliers
    K = [Ke intUStar; intUStar' 0];
    f = [Bqe; intU];

    nOfDOFUStar = refElem(pElemStar).nOfNodes*nOfComponents;
    indexEnd = indexIni + nOfDOFUStar - 1;

    % Elemental solution
    sol = K\f;
    uStar(indexIni:indexEnd) = sol(1:end-1);

    % Indexing
    indexIni = indexEnd + 1;
end
```

Fig. 21 Function `hdg_Poisson_LocalPostprocess` to perform a local, element-by-element, postprocess of the solution

in (27) features a saddle-point structure, as classical in the context of finite element approximations of incompressible flow problems [120]. In addition, since the HDG local problem is a purely Dirichlet boundary value problem, the constraint (23b) is introduced using a Lagrange multiplier ζ.

The structure of the symmetric indefinite matrix involved in the local problem is displayed on the left-hand side of Fig. 25. On the right-hand side, the blocks of the first and last columns are associated with the contribution of $\hat{\mathbf{u}}$ and $\boldsymbol{\rho}$, respectively, whereas the second column is related to the independent term of the equation. The figure reports the hybridisation stage in which the elemental matrices and vectors defined in (28) are computed for each element. The output of this computation is stored in the data structure `local` to be successively utilised in the solution of the element-by-element local problems.

Finally, `hdg_Stokes_ElementalMatrices` computes the elemental contribution to the matrix in the global problem (30) as reported in Fig. 26. It is worth noting that the variable `hdg.pureDirichlet` is utilised here to discriminate the construction of the global matrix of a purely Dirichlet boundary value problem. More precisely, besides the blocks $\widehat{\mathbf{K}}$, $\mathbf{G}$ and $\mathbf{G}^T$, also the vector $\mathbf{a}_{\bar{\rho}\rho}$ (i.e. `ArlExtra`) arising from the imposition of the global constraint (20) is taken into account. An extract of the `hdg_Stokes_ElementalMatrices` function computing such a vector is displayed in Fig. 27.

Remark 9 The assembly of the block matrix reported in Fig. 26 does not exploit the symmetry of the terms in Eq. (30) to its full potential. Indeed, the rationale of `HDG-lab` being educational, the matrix $\mathbf{G}$ is constructed by inserting the solution (28a) of the local problem into the

```
% Element computation ------------------------
[N, dNx, wXY, Xg] = ...
    gaussElemCartesianInfo(refElem(pElem), Xe);
Nw = bsxfun(@times, N, wXY');
NwN = Nw*N';

for iNsd = 1:nsd2
    vElem = iNsd:nsd2:ndofL;
    ALL(vElem,vElem) = -NwN;
end

sourceTerm = ...
    stokes_source(Xg,problemParams,matElem, ...
                  nsd,problemParams.example);

for iNsd = 1:nsd
    wElem = iNsd:nsd:ndofU;
    Apu(:,wElem) = dNx(:,:,iNsd)*Nw';
    fu(wElem) = Nw*sourceTerm(:,iNsd);
    for jNsd = 1:nsd
       zElem = (iNsd-1)*nsd+jNsd:nsd2:ndofL;
       ALu(zElem,wElem) = ...
            sqrt(nu)*dNx(:,:,jNsd)*Nw';
    end
end
```

Fig. 22 Extract of the `hdg_Stokes_ElementalMatrices` function that computes the element integrals of the HDG formulation for the Stokes problem

```
% Additional contributions if any of the ...
    faces features a slip BC
if isAnySlipFace
    AlL = zeros(ndofUhat,ndofL);
    Alu = zeros(ndofUhat,ndofU);
    Alp = zeros(ndofUhat,ndofP);
end
```

Fig. 23 Extract of the `hdg_Stokes_ElementalMatrices` function that initialises the elemental matrices associated with the slip boundaries in the global problem for the Stokes problem

```
% Flipping due to the different numbering ...
    of internal face nodes when seen from ...
    left or right element
ZLl = ZLl(:,indexFlip);
Zul = Zul(:,indexFlip);
Zpl = Zpl(:,indexFlip);
if isAnySlipFace
    AlL = AlL(indexFlip,:);
    Alu = Alu(indexFlip,:);
    Alp = Alp(indexFlip,:);
else
    AlL = ALl(:,indexFlip)';
    Alu = Aul(:,indexFlip)';
    Alp = Apl(:,indexFlip)';
end
All = All(indexFlip,indexFlip);
fl  = fl(indexFlip);
Arl = Arl(:,indexFlip);
```

Fig. 24 Extract of the `hdg_Stokes_ElementalMatrices` function that defines the elemental matrices of the global problem for the Stokes case, depending on the boundary conditions imposed

global problem (29), leading to the assembly of the elemental contributions

$$\mathbf{G}^e := \left[\mathbf{A}_{\hat{u}L}\ \mathbf{A}_{\hat{u}u}\ \mathbf{A}_{\hat{u}p}\ \mathbf{0}\right]_e \begin{Bmatrix} \mathbf{z}_L^\rho \\ \mathbf{z}_u^\rho \\ \mathbf{z}_p^\rho \\ \mathbf{z}_\zeta^\rho \end{Bmatrix}_e . \tag{37}$$

The interested reader can thus employ the provided code to numerically verify that the matrix $\mathbf{G}$ obtained from the assebly in Eq. (37) is the transpose of the one introduced in (31). The formal proof can be devised following the rationale described in [151].

9.4 Assembly of the Global System

As described in Sect. 4, the global problem for the Stokes equations features a saddle-point structure. Hence, the assembly of the global system presents major differences with respect to the Poisson case previously discussed. First, Fig. 28 reports the initialisation of the structures required to perform the assembly. It is worth recalling that the value of `hdg.rowsGlobalSystem` differs from the one of `hdg.columnsGlobalSystem` only if Dirichlet conditions are imposed on all boundaries. In this case, the additional constraint (20) is considered to guarantee the well-posedness of the problem.

The construction of the structures to perform the assembly of the global system is displayed in Fig. 29. According to the variable `hdg.pureDirichlet`, the above mentioned constraint is introduced as an extra line in the system.

Of course, the matrix arising from the operations displayed in Fig. 29 is rectangular. In order to retrieve a square matrix, an extra column is added to the matrix and the constraint is imposed via a Lagrange multiplier (Fig. 30).

9.5 Three Unknowns in the Local Problem

The structure of the code for the local problem in the Stokes case replicates the one presented in Fig. 20 for the Poisson equation and is thus omitted. The only difference lies in the computation of three variables in each element, namely the pressure $\mathbf{p}$, the velocity $\mathbf{u}$ and the gradient of velocity $\mathbf{L}$. An extract of the function `hdg_Stokes_LocalProblem` is displayed in Fig. 31, focusing on the operations in Eq. (28).

10 Visualisation

The use of a high-order functional approximation means that non-standard techniques are required to produce a reliable representation of the solution in each element. With the

```
% Elemental mapping ------------------------------------------------------------------------
Z = [ALL                 ALu             zeros(ndofL,ndofP)  zeros(ndofL,1);
     ALu'                Auu             Apu'                zeros(ndofU,1);
     zeros(ndofP,ndofL)  Apu             zeros(ndofP)                  Arp;
     zeros(1,ndofL)      zeros(1,ndofU)  Arp'                            0]\ ...
                                                        [ALl                fL  zeros(ndofL,1);
                                                         Aul                fu  zeros(ndofU,1);
                                                         Apl                fp  zeros(ndofP,1);
                                                         zeros(1,ndofUhat)  0                1];
```

Fig. 25 Extract of the `hdg_Stokes_ElementalMatrices` function that computes the matrices and vectors in Eq. (27)

```
% Elemental matrices
if isPureDirichlet
    Ke = [All + (AlL*ZLl + Alu*Zul + Alp*Zpl),  AlL*zLr + Alu*zur + Alp*zpr;
          Arl,                                  0;
          ArlExtra'*Zpl                         ArlExtra'*zpr];
    fe = [fl - (AlL*zLf + Alu*zuf + Alp*zpf);
          fn;
          ArlExtra'*zpf];
else
    Ke = [All + (AlL*ZLl + Alu*Zul + Alp*Zpl),  AlL*zLr + Alu*zur + Alp*zpr;
          Arl,                                  0];
    fe = [fl - (AlL*zLf + Alu*zuf + Alp*zpf);
          fn];
end
```

Fig. 26 Extract of the `hdg_Stokes_ElementalMatrices` function that computes the block matrix and vector in Eq. (30)

```
% Additional restriction for purely ...
    Dirichlet BC
ArlExtra = zeros(ndofP,1);

analyticalPressure = ...
    stokes_analyticalPressure(Xg,...
       problemParams,matElem,nsd,...
              problemParams.example);

% If purely Dirichlet BC
if isPureDirichlet
    ArlExtra = sum(Nw,2);
    intPressure = intPressure + ...
        sum(Nw*analyticalPressure,1);
end
```

Fig. 27 Extract of the `hdg_Stokes_ElementalMatrices` function that computes the vector $\mathbf{a}_{\bar{\rho}\rho}$ for the pressure constraint

```
% Initialisation
mat.i = zeros(1, hdg.rowsGlobalSystem);
mat.j = zeros(1, hdg.columnsGlobalSystem);
mat.Kij = zeros(1, hdg.nDOFglobal);
f = zeros(hdg.rowsGlobalSystem, 1);
```

Fig. 28 Extract of the `hdg_Stokes_Globalystem` function that initialises the structures required for the assembly of the global system

```
% Assembly
[indexGlobalFace, ~] = ...
        hdgElemToFaceIndex(mesh.indexTf,  ...
            refFace, hdg.faceInfo(iElem), ...
                 refElem(pElem).nOfFaces, ...
                              ctt.nOfComponents);
if hdg.pureDirichlet
   iBlock = [indexGlobalFace, ...
             hdg.nDOFglobal+iElem, ...
             hdg.nDOFglobal+mesh.nOfElements+1];
   jBlock = [indexGlobalFace, ...
             hdg.nDOFglobal+iElem];
else
   iBlock = [indexGlobalFace, ...
             hdg.nDOFglobal+iElem];
   jBlock = iBlock;
end

nI =numel(iBlock);
nJ =numel(jBlock);
currentIndex = indexIni + (1:nJ);
currentIndex2 = indexIni2 + (1:nI*nJ);
for i=1:nI
   mat.i(currentIndex) = iBlock(i);
   mat.j(currentIndex) = jBlock;
   currentIndex = currentIndex + nJ;
end
mat.Kij(currentIndex2) = ...
                         reshape(Ke', 1, nI*nJ);
```

Fig. 29 Extract of the `hdg_Stokes_Globalystem` function that constructs the structures required for the assembly of the global system

```
if hdg.pureDirichlet
    f(hdg.rowsGlobalSystem) = ...
        -f(hdg.rowsGlobalSystem)+intPressure;
    K = [K, [K(end,:)'; 0]];
end
```

Fig. 30 Extract of the `hdg_Stokes_Globalystem` function to impose the constraint in the global system for purely Dirichlet boundary value problems

increased popularity of high-order methods in recent years, different strategies to efficiently display high-order solutions have been proposed [188, 198, 220]. The `HDGlab` provides the capability to accurately display high-order solutions, including curved isoparametric elements.

Three data structures are employed in the visualisation. The data structure `plotOpts` is used to collect all the user defined options available for the visualisation. It contains the following information:

- `resolution`: Takes value 1 for a faster but less accurate representation of high-order solutions and value 2 for a slower but more accurate representation of high-order solutions.
- `fieldsWithMesh`: Plots the solution whilst displaying the mesh.
- `fieldsWithNodes`: Plots the solution whilst displaying the nodes.
- `componentsToPlot`: A user-defined vector containing the components of the solution to be represented.
- `alphaFace`: Sets the transparency between 0 and 1.

In addition, for the Stokes equation, two problem-specific options are available in the data structure `plotOpts`:

- `componentsU`: A boolean variable allowing the predefined visualisation of all the components of the velocity vector. This functionality relies on the definition of the vector `componentsToPlot`.
- `moduleU`: A boolean variable allowing the visualisation of the module of the velocity.

The data structure `postproc` is provided for triangular and tetrahedral elements with equally-spaced and Fekete nodal sets in the directory `dat/postprocess`. In two dimensions, the data structure `postproc` contains the following information:

- `nOfNodesPlot`: Number of nodes used to display the high-order solution in each element.
- `nodesPlot`: Array of dimension $\texttt{nOfNodesPlot} \times 2$ containing the coordinates of the nodes, in the reference element, used to display the high-order solution.
- `nSubElemsOnePlot`: Number of subelements used to display the high-order solution in each element.
- `connecNodesPlot`: Array of dimension $\texttt{nSubElemsOnePlot} \times 3$ accounting for the connectivity of the submesh used to display the high-order solution in each element.
- `nOfEdges`: Number of edges of the element.
- `edgeNodesSplit`: Array of dimension $5\texttt{r} \times \texttt{nOfEdges}$, where `r` is the resolution selected by the user in the data structure `plotOpts`. The i-th column contains the local number of the list of `nodesPlot` that belong to the i-th edge.
- `elem`: Array of dimension $1 \times \texttt{p}_{\max}$, where $\texttt{p}_{\max}$ is the maximum degree of approximation used in all the elements. The component `p` of `elem` contains a field called `N`, of dimension $\texttt{nOfNodesPlot} \times \texttt{p}(\texttt{p}+1)/2$, that stores the value of the shape functions of order `p` at the positions given by `nodesPlot`. This information is used to interpolate the solution at the nodes of the submesh, providing a more accurate representation of the high-order solution.
- `face`: Array of dimension $1 \times \texttt{p}_{\max}$. The component `p` of `elem` contains a field called `N`, of dimension $\texttt{nOfNodesPlot} \times (\texttt{p}+1)$, that stores the value of the shape functions of order `p` at the positions of an edge. This information is used to interpolate the solution at the edges of the submesh.

The submeshes used for a triangular element with `resolution=1` and `resolution=2` are displayed in Fig. 32. To illustrate the effect of the user-defined parameter `resolution` on the visualisation, Fig. 33 depicts the shape function associated to the fourth node of the reference quadratic triangular element, shown in Fig. 8, using `resolution=1` and `resolution=2`.

```
p(indexPIni:indexPEnd) = local(iElem).Zpl*uHat(indexGlobalFace) + local(iElem).zpr*rho(iElem) ...
    + local(iElem).zpf;
u(indexUIni:indexUEnd) = local(iElem).Zul*uHat(indexGlobalFace) + local(iElem).zur*rho(iElem) ...
    + local(iElem).zuf;
L(indexLIni:indexLEnd) = local(iElem).ZLl*uHat(indexGlobalFace) + local(iElem).zLr*rho(iElem) ...
    + local(iElem).zLf;
```

Fig. 31 Extract of the `hdg_Stokes_LocalProblem` function where the elemental values of the primal and mixed unknowns are computed

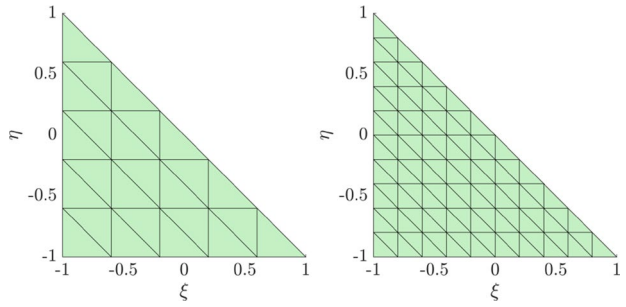


Fig. 32 Submesh of the reference element used to provide an accurate representation of high-order solutions in each element. The left picture corresponds to `resolution=1` and the right picture to `resolution=2`

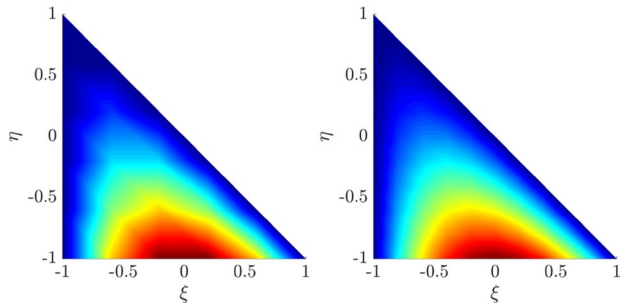


Fig. 33 Shape function of the fourth node of a quadratic triangular element using `resolution=1` (left) and `resolution=2` (right)

In three dimensions the data structure `postproc` contains the following information:

- `Face`: Structure that contains:
  - `nElemPlot`: Number of subelements used to display the high-order solution in each element face.
  - `connecPlot`: Array of dimension `nElemPlot` $\times 3$ accounting for the connectivity of the submesh used to display the high-order solution in each element face.
  - `nNodesPlot`: Number of nodes used to display the high-order solution on each element face. It is given by $(5\mathtt{r}+1)(5\mathtt{r}+2)/2$, where `r` is the resolution selected by the user in the data structure `plotOpts`.
  - `edgeNodesPlot`: Array of dimension $1 \times (15\mathtt{r}+1)$, where `r` is the resolution selected by the user in the data structure `plotOpts`. It contains the local number of the face nodes that belong to the edges of the face.
- `Elem`: Array of dimension $1 \times \mathtt{p}_{\max}$, where $\mathtt{p}_{\max}$ is the maximum degree of approximation used in all the elements. The component `p` of `Elem` contains:
  - `face`: Array of dimension 1×4 where the i-th component contains the local number of the vertices on the i-th face and the value of the element shape functions of order `p` at the nodal distribution used for plotting the solution on the i-th face.
  - `coord`: Array of dimension $\mathtt{p}(\mathtt{p}+1)(\mathtt{p}+2)/6$ that contains the nodal distribution on the reference tetrahedral for an approximation of degree `p`.
- `faceVertices`: Array of dimension 4×3. The i-th row contains the local number of the vertices on the i-th face.

The third data structure utilised during the postprocess stage is called `visual` and it is built by the function `buildSubmeshPostprocess2D` in two dimensions and by `buildSubmeshPostprocess3D` in three dimensions. This data structure contains the following information:

- `X`: Physical coordinates of all the nodes of the submesh used to display the high-order solution.
- `T`: Connectivities of all the subelements used to display the high-order solution.
- `Xnodes`: Physical coordinates of all the vertices of the mesh. This field is only used if the user sets `fieldsWithNodes=1` in the data structure `plotOpts`.
- `edges`: Structure containing the list of nodes of the submesh that form the high-order representation of the physical edges of the mesh.

In a separate function, the data structure is updated by adding the field `U` that contains the interpolated values of the solution on the submesh used to display the high-order solution. This action is performed by the function called `interpolateSolutionPostprocess2D` and `interpolateSolutionPostprocess3D` in two and three dimensions respectively.

It is worth noting that the function that creates the data structure `visual` is independent on the field to be represented and, therefore, it is only called once. Instead, the second function that updates the data structure `visual` with the field `U` depends upon the field to be represented. Therefore, several calls can be made to the function updating `visual` without the need to build the submesh again. It is also important to note that the function that updates the data structure `visual` with the field `U` accepts elemental and nodal fields.

Once all the information is available in the data structure `visual`, the function `postprocessField2D` is used to plot the high-order solution.

In three dimensions there is an extra option available that consists of representing the solution only in a region of the computational domain. The user can set the value of a string, called `conditionPlot`, that specifies a region in the physical space. Before `visual` is computed, the function `selectFacesToPlot3D` computes the list of faces in the computational mesh that satisfy the condition given by `conditionPlot`. Then, the submesh and interpolation of the solution is only performed over the faces that satisfy the condition specified by the user.

Finally, the visualisation function also accounts for the need to represent the superconvergent solution obtained after the local postprocess described in Sects. 3.4 and 4.4. To simplify the implementation, the visualisation builds a `mesh` data structure where the degree of approximation in each element is the degree used for the computation plus one. With this information, the same functions used to display the high-order primal solution can be used to postprocess the higher order superconvergent solution.

11 Numerical Examples

In this section, several numerical examples showing the capabilities of the `HDGlab` solvers for the Poisson and Stokes equations are presented. As mentioned in Sects. 3 and 4, the choice of the stabilisation parameter τ is critical for the accuracy of the HDG approximation. For the examples involving the Poisson equation, the definition $\tau = c_p \kappa/\ell$ is considered, where ℓ is a characteristic length of the domain and c_p a scaling factor selected equal to 1 [175]. Following [151], the stabilisation for the Stokes cases is defined as $\tau = c_s \nu/\ell$, ℓ the scaling factor being $c_s = 3$.

11.1 Optimal Convergence Properties

The optimal convergence properties of the proposed HDG implementation are presented for the Stokes flow, using test cases with analytical solution, in two and three dimensions. Uniform meshes of triangular and tetrahedral elements with Fekete nodal sets are utilised.

First, the two-dimensional Wang flow [264] in the unit square domain $\Omega = [0,1]^2$ is considered. The analytical velocity field is

$$\boldsymbol{u}(\boldsymbol{x}) = \left\{ \begin{array}{l} 2ax_2 - b\lambda\cos(\lambda x_1)\exp\{-\lambda x_2\} \\ b\lambda\sin(\lambda x_1)\exp\{-\lambda x_2\} \end{array} \right\}, \tag{38}$$

whereas the pressure field is uniformly zero in Ω. The coefficients a, b and λ in (38) are selected such that $a = b = 1$ and $\lambda = 10$ and the kinematic viscosity ν is set to 1. The source term $\boldsymbol{s}$ and the boundary conditions are computed starting from the analytical solution above. More precisely, a pseudo-traction $\boldsymbol{g}$ is applied on the bottom surface $\Gamma_N := \{(x_1, x_2) \in \Omega \mid x_2 = 0\}$ and Dirichlet data $\boldsymbol{u}_D$ are imposed on the remaining boundaries $\Gamma_D = \partial\Omega \backslash \Gamma_N$.

Figure 34 displays the convergence history of the relative error, measured in the $\mathcal{L}_2(\Omega)$ norm, of the primal, mixed and postprocessed variables as a function of the characteristic mesh size. Optimal convergence of order p + 1 is observed for velocity, $\boldsymbol{u}$, pressure, p, and gradient of velocity, $\boldsymbol{L}$, whereas superconvergence of order p + 2 is achieved by the postprocessed velocity $\boldsymbol{u}_\star$.

The following example involves a three-dimensional Stokes flow in the unit cube $\Omega = [0,1]^3$, with the following manufactured solution

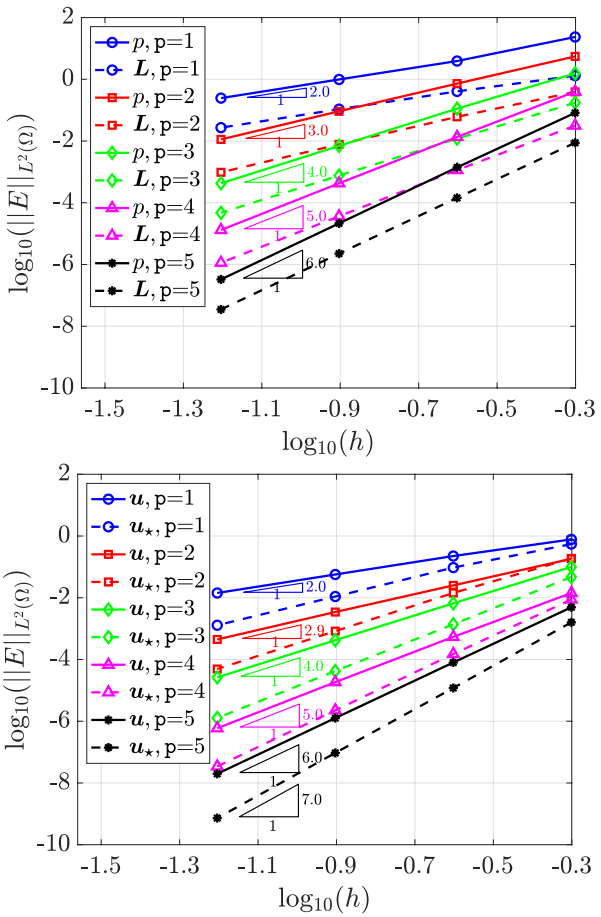


Fig. 34 Two-dimensional Wang flow. Convergence of the $\mathcal{L}_2(\Omega)$ error of pressure, p, mixed variable, $\boldsymbol{L}$ (top), primal, $\boldsymbol{u}$, and postprocessed, $\boldsymbol{u}_\star$, velocities (bottom) as a function of the characteristic mesh size h for polynomial degree of approxiomation p = 1, … , 5

$$\boldsymbol{u}(\boldsymbol{x}) = \left\{ \begin{array}{l} n + (x_3 - x_2)\sin(x_1 - n) \\ m - x_2 \left[\left(x_3 - \frac{1}{2} x_2 \right) \cos(x_1 - n) \right. \\ \qquad \left. + \left(x_1 - \frac{1}{2} x_2 \right) \cos(x_3 - n) \right] \\ n + (x_1 - x_2)\sin(x_3 - n) \end{array} \right\}, \tag{39}$$

$$p(\boldsymbol{x}) = x_1(1 - x_1) + x_2(1 - x_2) + x_3(1 - x_3), \tag{40}$$

where $m = 1$ and $n = \frac{1}{2}$. The kinematic viscosity is set to $\nu = 1$, a Neumann datum is imposed on the boundary $\Gamma_{_N} := \{(x_1, x_2, x_3) \in \Omega \mid x_3 = 0\}$, whereas Dirichlet conditions are prescribed on $\Gamma_{_D} = \partial\Omega \backslash \Gamma_{_N}$.

The optimal convergence of order $\mathrm{p} + 1$ of the relative $\mathcal{L}_2(\Omega)$ error for velocity, pressure and gradient of velocity and the superconvergence of the postprocessed velocity are confirmed in the 3D case by the results in Fig. 35.

11.2 High-Order Curved Meshes

The coaxial Couette flow [64] is considered to show the optimal convergence properties of the `HDGlab` solver using a high-order isoparametric approximation in a domain featuring curved boundaries.

This test consists of an incompressible viscous flow within two coaxial circular cylinders of infinite length and radius $R_{\text{int}} = 1$ and $R_{\text{ext}} = 5$, respectively. The computational domain is defined as a section of the 3D cylinders, that is, $\Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid R_{\text{int}} \leq r \leq R_{\text{ext}}\}$, where $r := \sqrt{x_1^2 + x_2^2}$ is the distance to the axis of the cylinders. Dirichlet boundary conditions enforcing the value of the angular velocities $\omega_{\text{int}} = 0$ and $\omega_{\text{ext}} = 1/R_{\text{ext}}$ are imposed on the internal and external boundary, respectively. The analytical expression of the azimuthal component of the velocity is

$$u_\phi = \frac{\omega_{\text{ext}} R_{\text{ext}}^2 - \omega_{\text{int}} R_{\text{int}}^2}{R_{\text{ext}}^2 - R_{\text{int}}^2} r - \frac{(\omega_{\text{ext}} - \omega_{\text{int}}) R_{\text{ext}}^2 R_{\text{int}}^2}{R_{\text{ext}}^2 - R_{\text{int}}^2} \frac{1}{r}. \tag{41}$$

Of course, being a purely Dirichlet boundary value problem, the constraint on the mean value of pressure is introduced to enforce the field to be uniformly equal to 1 in the domain.

A set of high-order uniformly refined meshes with Fekete nodal distribution is constructed using the strategy described in [212]. Figure 36 displays the first level of mesh refinement featuring 128 triangular elements of polynomial degree 3 and the module of the computed velocity.

The convergence of the relative error, measured in the $\mathcal{L}_2(\Omega)$ norm, of the primal, mixed and postprocessed variables is reported in Fig. 37 as a function of the characteristic mesh size. Optimal convergence of the primal and mixed variables and superconvergence of the postprocessed variable is achieved also in presence of high-order curved meshes.

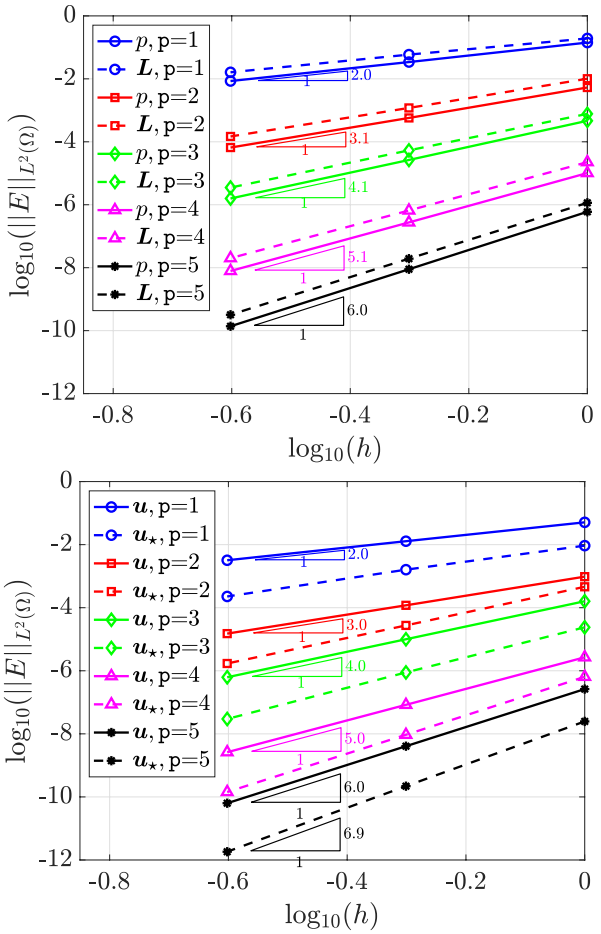


Fig. 35 Three-dimensional manufactured Stokes flow. Convergence of the $\mathcal{L}_2(\Omega)$ error of pressure, p, mixed variable, $\boldsymbol{L}$ (top), primal, $\boldsymbol{u}$, and postprocessed, $\boldsymbol{u}_\star$, velocities (bottom) as a function of the characteristic mesh size h for polynomial degree of approxiomation $\mathrm{p} = 1, \ldots, 5$

11.3 Non-uniform Degree of Approximation

In this section, the flexibility of `HDGlab` to devise a non-uniform polynomial degree approximation in the domain is presented. This case, inspired by the study on micromixers in [131], consists of the flow in a microchannel with five obstacles. The problem setup features a parabolic inlet velocity profile and homogeneous Dirichlet and Neumann conditions on the top/bottom walls and on the outlet, respectively.

The channel has dimensions $[0, 6.6] \times [-0.5, 0.5]$ and the obstacles, attached to the top and bottom walls have thickness 0.2 and height 0.5. A mesh with local element size ranging between 0.08 and 0.19 is generated without any specific *a priori* refinement. It is worth noting that

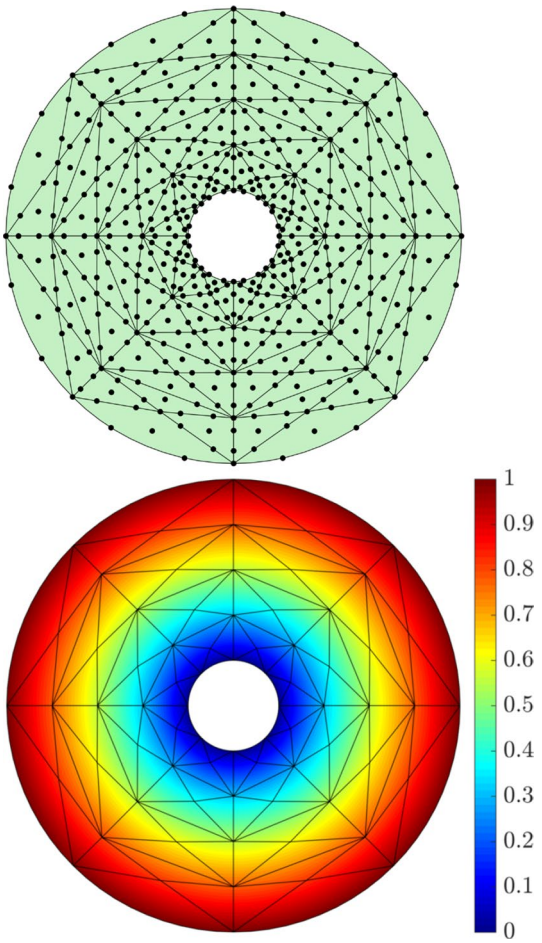


Fig. 36 First level of refinement of the third-order mesh used for the convergence study of the two-dimensional Couette flow (top) and module of the computed velocity (bottom)

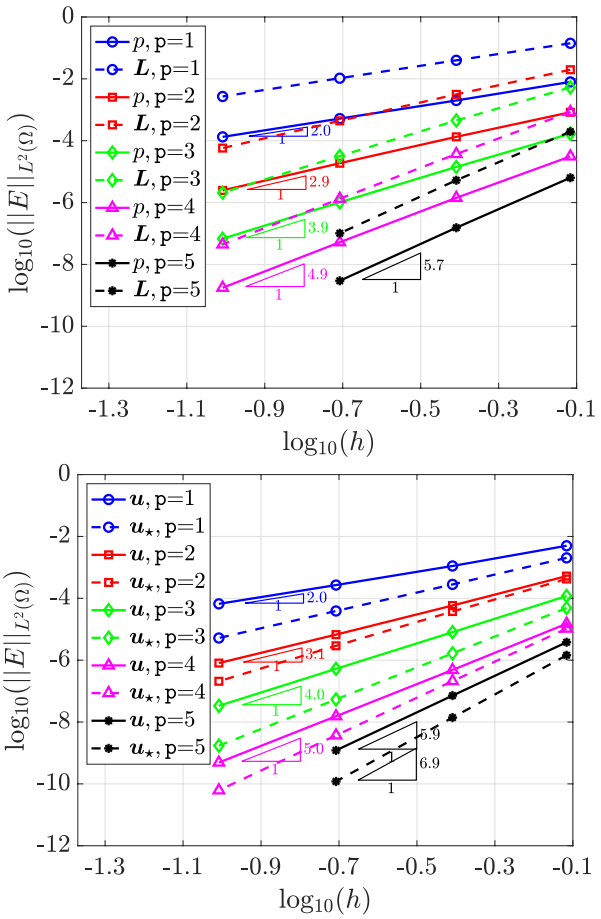


Fig. 37 Two-dimensional Couette flow. Convergence of the $\mathcal{L}_2(\Omega)$ error of pressure, p, mixed variable, $\boldsymbol{L}$ (top), primal, $\boldsymbol{u}$, and post-processed, $\boldsymbol{u}_\star$, velocities (bottom) as a function of the characteristic mesh size h for polynomial degree of approxiomation p = 1, … , 5

only two mesh elements are defined along the thickness of the obstacles.

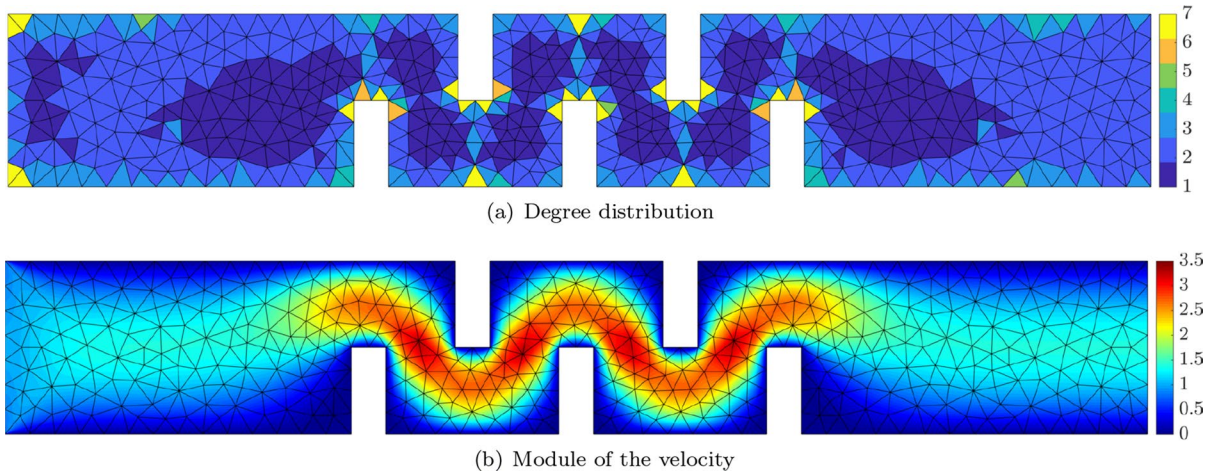
To capture the complex flow features among the obstacles, a high-order non-uniform polynomial degree distribution is generated following the adaptivity strategy described in [247]. The resulting degree of approximation in each element is displayed in Fig. 38. High-order polynomials are employed in correspondance of the tip of the obstacles where localised flow features appear, whereas low-order approximations are utilised in region further away.

The module of the velocity on the described mesh is also reported in Fig. 38. It is worth noting that the mesh structure featuring the non-uniform polynomial approximation is provided as a datum for this test case. The corresponding simulation is performed seamlessly in `HDGlab` and no specific intervention is required to the user.

11.4 Stokes Flow Past a Sphere

Finally, the Stokes flow past a sphere is considered. The domain $\Omega = [-H, L] \times [-H, H] \times [-H, H] \backslash \mathcal{B}_{\mathbf{0},1}$ is defined, with $H = 5$, $L = 10$ and $\mathcal{B}_{\mathbf{0},1}$ being the sphere of radius 1 centred in the point (0, 0, 0). Exploiting the symmetry of the problem, only a quarter of the domain is meshed and slip conditions are imposed on the corresponding symmetry planes. Homogeneous Neumann and no-slip conditions are applied on the outlet and on the surface of the sphere, respectively. On the inlet and on the remaining lateral and top planes, a Dirichlet boundary conditions with the analytical velocity is enforced.

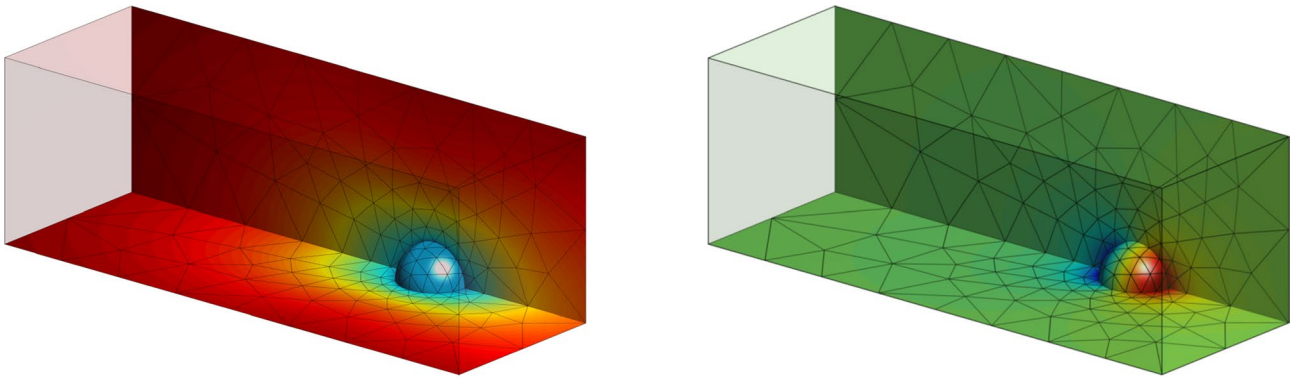
A high-order mesh featuring 1,036 tetrahedral elements is generated via the solid mechanics analogy described in [212, 269]. Figure 39 displays the module of the velocity and the pressure field computed using an isoparametric approximation of degree 6.



(a) Degree distribution

(b) Module of the velocity

Fig. 38 **a** Mesh and degree of approximation and **b** module of the velocity for the flow in a microchannel with obstacles using non-uniform polynomial degree p between 1 and 7



(a) Module of the velocity

(b) Pressure

Fig. 39 **a** Module of the velocity and **b** pressure field of the external flow past a quarter of a sphere using polynomial degree $p = 6$

It is worth noting that the computations in Sects. 11.1, 11.2, 11.3 and 11.4 are performed using a unique `HDGlab` solver for the Stokes equations, independently on the number of spatial dimensions of the problem under analysis. Indeed, `HDGlab` provides a seamless implementation of the HDG method, in which all relevant information is extracted from the mesh structure and the user is required to specify only the physical parameters and the boundary conditions to setup a test case.

11.5 Applications of the Poisson Solver

The next example, taken from [187], shows the solution of an electrostatic problem governed by the Poisson equation in three dimensions. The domain of interest corresponds to the exterior of 11 conducting spheres and it is discretised with a mesh of 35,895 quadratic tetrahedral elements, as shown in Fig. 40. This figure shows one of the implemented capabilities of the postprocessing library to display the faces corresponding to the exterior and interior faces of the mesh separately, with different colour and transparencies in each case. The first plot in Fig. 40 is produced by selecting the faces corresponding to the far field boundary and using a transparency. In a second phase, the exterior faces corresponding to the conducting spheres are displayed with no transparency. It is worth noting that the far field boundary and the boundary corresponding to the conducting spheres could be distinguished using either the boundary condition flag or simply imposing a condition on the faces to be displayed. The second plot of Fig. 40 is also obtained in two stages. First, the interior faces satisfying a condition corresponding to a positive x_2 coordinate are displayed with a transparency. Second, the exterior faces corresponding to the conducting spheres is displayed with no transparency. When representing the interior and

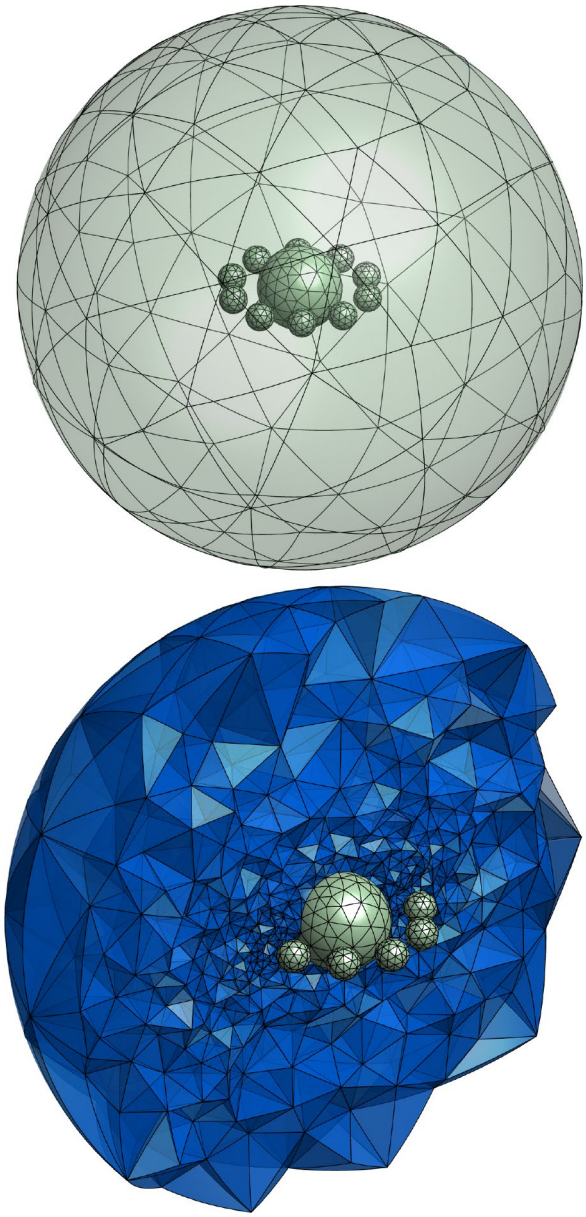


Fig. 40 Two views of the quadratic tetrahedral mesh used to solve the electrostatic problem

exterior faces, a constant element field is used to assign different colours and aid the visualisation.

Dirichlet boundary conditions are considered in the whole boundary of the computational domain. A positive electrostatic potential of magnitude 5 is imposed on the central sphere and five of the surrounding spheres, whereas a negative potential of magnitude −5 is imposed on the remaining spheres. On the outer boundary a zero potential is imposed. Figure 41 shows the 11 conducting spheres coloured according to the boundary condition imposed. This figure is produced by using the boundary condition flag to select the first set and the second set of spheres separately. After solving

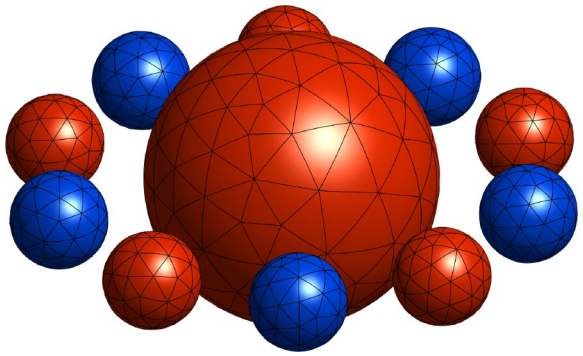


Fig. 41 The 11 conducting spheres coloured according to the boundary condition imposed. Red colour is used for the spheres where a positive potential is imposed, whereas blue is used for the spheres where a negative potential is imposed

the problem with HDGlab, not only the primal variable corresponding to the electrostatic potential is obtained but also its gradient, which corresponds to the electric field in this example. Figure 42 shows the magnitude of the electric field on the surface of the 11 spheres.

The following example involves the solution of a heat transfer problem in a three dimensional mechanical component. The domain is discretised with a mesh of 6465 elements with $p = 4$, as illustrated in Fig. 43.

The first plot in Fig. 43 includes the high-order nodal distribution and the second plot in Fig. 43 shows the possibilities offered by the postprocessing library provided within HDGlab. In fact, it shows an extra functionality that can be added to display the intersection curves of the underlying CAD geometry when the user have access to this data.

Dirichlet and Neumann boundary conditions are imposed on the blue and red portions of the boundary, respectively, as shown in Fig. 44. In the Dirichlet part of the boundary a temperature equal to 10 is imposed, whereas the Neumann boundary condition is homogeneous, imposing that part of the boundary is perfectly insulated. The temperature field

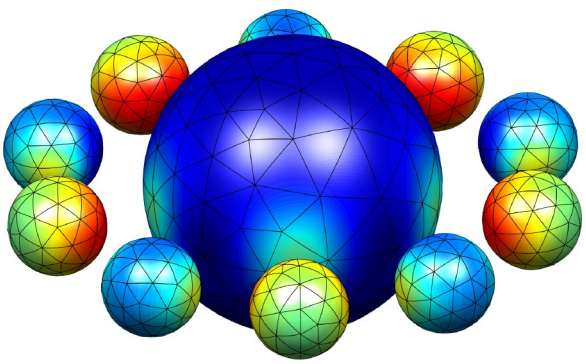


Fig. 42 Magnitude of the electric field on the surface of the 11 conducting spheres

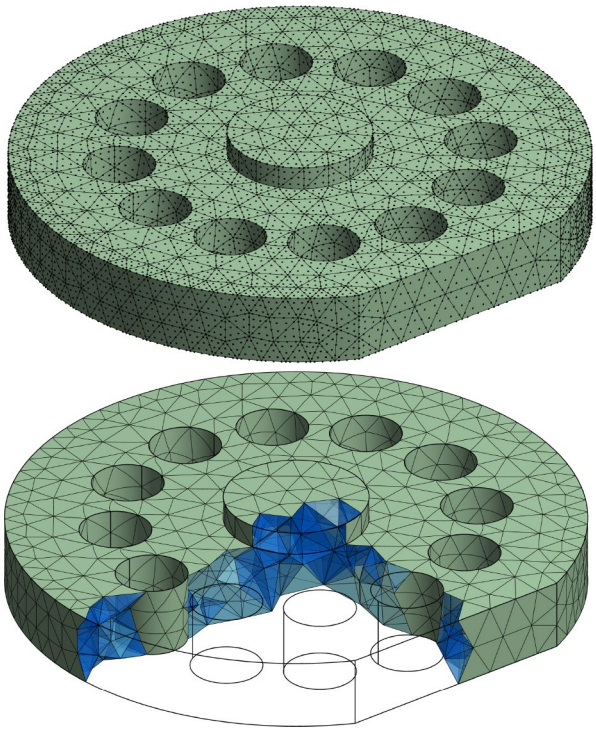


Fig. 43 Two views of the fourth order tetrahedral mesh used to solve the heat transfer problem

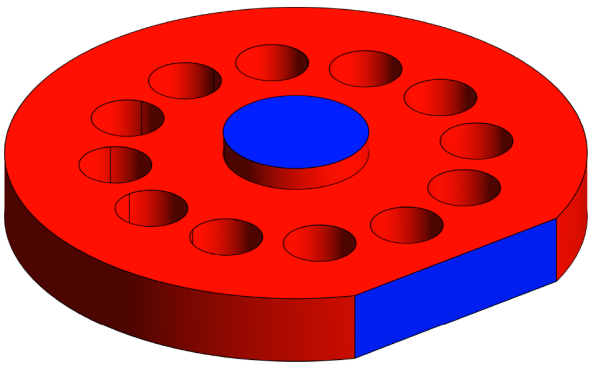


Fig. 44 The mechanical component coloured according to the boundary condition imposed. Red colour denotes a homogeneous Neumann boundary condition and the blue colour a Dirichlet boundary condition

obtained after solving the problem with `HDGlab` is shown in Fig. 45.

The last example considers the computation of the potential flow past a generic unmanned aerial vehicle (UAV). The domain is discretised using 101,923 tetrahedra of degree

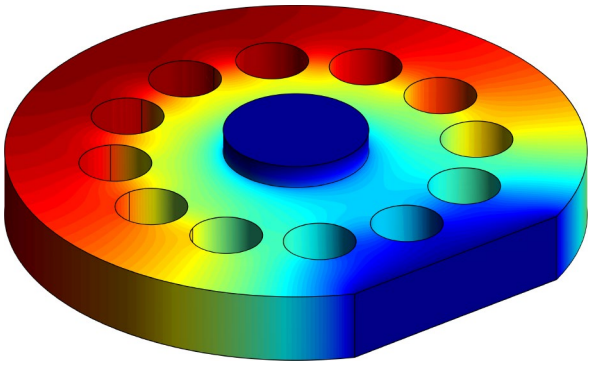


Fig. 45 Temperature distribution on the surface of the mechanical component

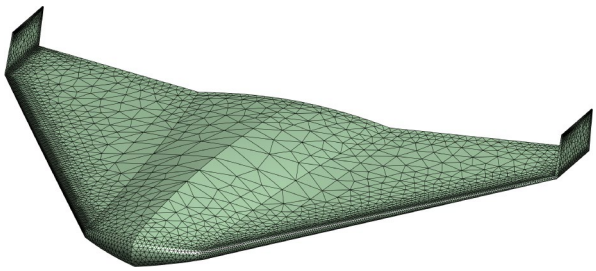


Fig. 46 Surface mesh of a generic UAV

$p = 2$, as represented in Fig. 46. A Neumann boundary condition, corresponding to a unit velocity, is imposed on the inflow part of the boundary and a Dirichlet boundary condition corresponding to zero potential on the outflow. On the rest of the boundary a homogeneous Neumann boundary condition is enforced to represent a physical wall. After solving the problem with `HDGlab`, the flow potential and the velocity field are obtained. Figure 47 shows the magnitude of the velocity field on the surface of the UAV and the pressure field, computed using the Bernoulli equation.

This last example shows the applicability of the developed `HDGlab` to problems involving complex geometries, where the resulting global system has more than one million of equations. Despite the code was not developed with computational efficiency in mind, it still enables the interested users to solve relatively large problems and, with some additional improvements in terms of performance, it can be used for larger three dimensional problems.

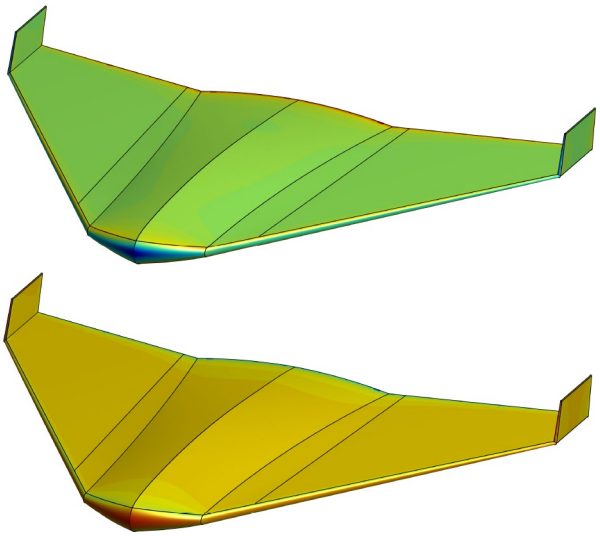


Fig. 47 Magnitude of the velocity field and pressure field on the surface of a generic UAV

12 Concluding Remarks

An open-source Matlab implementation of the HDG method for elliptic problems has been presented, the so-called HDGlab. The code is capable of solving problems governed by the Poisson and Stokes equations using high-order simplicial elements, including curved elements, by means of an isoparametric formulation.

HDGlab provides a suitable environment to those interested in the programming of hybrid methods in general and the HDG method in particular. The paper describes the HDG formulation employed for the Poisson and Stokes solvers and provides a very detailed description of the code, with particular emphasis in the data structures utilised. The presentation of the code involves the preprocess, computation and postprocess stages, and includes detailed explanations of the most relevant functions. A set of examples is presented to illustrate the use and the potential of HDGlab. The examples go from simple test cases with a known analytical solution, used to demonstrate the numerical properties of the HDG method, to more complex ones such as the computation of the potential flow around a UAV using high-order curved meshes.

HDGlab is available as an open-source software, released under the terms of the GNU General Public License version 3.0 or any later version (https://www.gnu.org/licenses) and is freely available from the repository: https://git.lacan.upc.edu/hybridLab/HDGlab.

Acknowledgements This work was partially supported by the Spanish Ministry of Economy and Competitiveness (Grant Number: DPI2017-85139-C2-2-R). M.G. and A.H. are also grateful for the support provided by the Spanish Ministry of Economy and Competitiveness through the Severo Ochoa programme for centres of excellence in RTD (Grant Number: CEX2018-000797-S) and the Generalitat de Catalunya (Grant Number: 2017-SGR-1278). R.S. also acknowledges the support of the Engineering and Physical Sciences Research Council (Grant Number: EP/T009071/1).

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Appendix 1: Matrices and Vectors for the Poisson Solver

In this appendix, the expressions of the matrices and vectors appearing in the discrete form of the HDG local and global problems for the Poisson equation are presented.

An isoparametric formulation is considered and the coordinates $\boldsymbol{\xi} = \{\xi_1, \dots, \xi_{\mathrm{n_{sd}}}\}^T$ of a reference element $\widetilde{\Omega}(\boldsymbol{\xi})$ are mapped to the coordinates $\boldsymbol{x} = \{x_1, \dots, x_{\mathrm{n_{sd}}}\}^T$ of the local element $\Omega(\boldsymbol{x})$ by the transformation

$$\boldsymbol{x}(\boldsymbol{\xi}) = \sum_{i=1}^{\mathrm{n_{en}}} N_i(\boldsymbol{\xi})\boldsymbol{x}_i,$$

where $\{\boldsymbol{x}_i\}_{i=1,\dots,\mathrm{n_{en}}}$ denotes the vector of the nodal coordinates of the element. Hence, the isoparametric approximations introduced in (8) are defined in a reference element $\widetilde{\Omega}(\boldsymbol{\xi})$ for u and $\boldsymbol{q}$ and on a reference face $\widetilde{\Gamma}(\boldsymbol{\eta})$, $\boldsymbol{\eta} = \{\eta_1, \dots, \eta_{\mathrm{n_{sd}}-1}\}^T$ for $\hat{u}$, the corresponding shape functions being $N(\boldsymbol{\xi})$ and $\hat{N}(\boldsymbol{\eta})$, respectively.

For the sake of readability, introduce the compact forms of the shape functions and their derivatives, namely

$$\boldsymbol{\mathcal{N}} := \begin{bmatrix} N_1 & N_2 & \dots & N_{\mathrm{n_{en}}} \end{bmatrix}^T, \tag{42a}$$

$$\hat{\boldsymbol{\mathcal{N}}} := \begin{bmatrix} \hat{N}_1 & \hat{N}_2 & \dots & \hat{N}_{\mathrm{n_{fn}}} \end{bmatrix}^T, \tag{42b}$$

$$\boldsymbol{\mathcal{N}}_n := \begin{bmatrix} N_1\boldsymbol{n} & N_2\boldsymbol{n} & \dots & N_{\mathrm{n_{en}}}\boldsymbol{n} \end{bmatrix}^T, \tag{42c}$$

$$\boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}} := \left[N_1 \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; N_2 \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; \dots \; N_{\mathtt{n}_{\mathtt{en}}} \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \right]^T, \tag{42d}$$

$$\boldsymbol{\mathcal{Q}} := \left[(\mathbf{J}^{-1}\boldsymbol{\nabla} N_1)^T \; (\mathbf{J}^{-1}\boldsymbol{\nabla} N_2)^T \; \dots \; (\mathbf{J}^{-1}\boldsymbol{\nabla} N_{\mathtt{n}_{\mathtt{en}}})^T \right]^T, \tag{42e}$$

where $\boldsymbol{n}$ denotes the outward unit normal vector to a face and $\mathbf{J}$ is the Jacobian of the isoparametric mapping.

The solution of the HDG local problem (9) involves the matrices and vectors

$$\left[\mathbf{A}_{uu}\right]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\xi^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^T(\xi^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}},$$

$$\left[\mathbf{A}_{uq}\right]_e := \sqrt{\kappa} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\xi^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{\mathcal{Q}}^T(\xi^{\mathtt{e}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}},$$

$$\left[\mathbf{A}_{qq}\right]_e := - \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}}(\xi^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^T_{\mathtt{n}_{\mathtt{sd}}}(\xi^{\mathtt{e}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}},$$

$$\left[\mathbf{A}_{u\hat{u}}\right]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\xi^{\mathtt{f}}_{\mathtt{g}}) \widehat{\boldsymbol{\mathcal{N}}}^T(\xi^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \left(1 - \chi^{\mathtt{f}}_{D}\right),$$

$$\left[\mathbf{A}_{q\hat{u}}\right]_e := \sqrt{\kappa} \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_n(\xi^{\mathtt{f}}_{\mathtt{g}}) \widehat{\boldsymbol{\mathcal{N}}}^T(\xi^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \left(1 - \chi^{\mathtt{f}}_{D}\right),$$

$$\begin{aligned} \left[\mathbf{f}_u\right]_e := & \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\xi^{\mathtt{e}}_{\mathtt{g}}) s\left(\boldsymbol{x}(\xi^{\mathtt{e}}_{\mathtt{g}})\right) |\mathbf{J}(\xi^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}} \\ & + \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\xi^{\mathtt{f}}_{\mathtt{g}}) u_{D}\left(\boldsymbol{x}(\xi^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{D}, \end{aligned}$$

$$\left[\mathbf{f}_q\right]_e := \sqrt{\kappa} \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_n(\xi^{\mathtt{f}}_{\mathtt{g}}) u_{D}\left(\boldsymbol{x}(\xi^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{D},$$

where $\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}$ denotes the number of faces of the element Ω_e, $\xi^{\mathtt{e}}_{\mathtt{g}}$ are the $\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}$ integration points defined on the reference element and $\xi^{\mathtt{f}}_{\mathtt{g}}$ are the $\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}$ integration points of the reference face. The corresponding weights for the integration points are denoted by $w^{\mathtt{e}}_{\mathtt{g}}$ and $w^{\mathtt{f}}_{\mathtt{g}}$, respectively. In addition, the indicator function $\chi^{\mathtt{f}}_{\square}$ is defined as

$$\chi^{\mathtt{f}}_{\square} := \begin{cases} 1 & \text{if face } \mathtt{f} \text{ belongs to } \Gamma_{\square}, \\ 0 & \text{otherwise.} \end{cases} \tag{43}$$

Similarly, for the HDG global problem (11) the following matrix and vector

$$\left[\mathbf{A}_{\hat{u}\hat{u}}\right]_e := - \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \widehat{\boldsymbol{\mathcal{N}}}(\xi^{\mathtt{f}}_{\mathtt{g}}) \widehat{\boldsymbol{\mathcal{N}}}^T(\xi^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \left(1 - \chi^{\mathtt{f}}_{D}\right),$$

$$\left[\mathbf{f}_{\hat{u}}\right]_e := - \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \widehat{\boldsymbol{\mathcal{N}}}(\xi^{\mathtt{f}}_{\mathtt{g}}) g\left(\boldsymbol{x}(\xi^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\xi^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{N},$$

are defined.

To describe the terms appearing in the HDG post-process (17), the compact forms

$$\boldsymbol{\mathcal{N}}^{\star} := \left[N^{\star}_1 \; N^{\star}_2 \; \dots \; N^{\star}_{\mathtt{n}^{\star}_{\mathtt{en}}} \right]^T, \tag{44a}$$

$$\boldsymbol{\mathcal{N}}^{\star}_{\mathtt{n}_{\mathtt{sd}}} := \left[N^{\star}_1 \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; N^{\star}_2 \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; \dots \; N^{\star}_{\mathtt{n}^{\star}_{\mathtt{en}}} \mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \right]^T, \tag{44b}$$

$$\boldsymbol{\mathcal{Q}}^{\star} := \left[(\mathbf{J}^{-1}_{\star}\boldsymbol{\nabla} N^{\star}_1)^T \; (\mathbf{J}^{-1}_{\star}\boldsymbol{\nabla} N^{\star}_2)^T \; \dots \; (\mathbf{J}^{-1}_{\star}\boldsymbol{\nabla} N^{\star}_{\mathtt{n}^{\star}_{\mathtt{en}}})^T \right]^T, \tag{44c}$$

are introduced, where $\{N^{\star}_i\}_{i=1,\dots,\mathtt{n}^{\star}_{\mathtt{en}}}$ denote the shape functions for the discretisation of $u_{\star}$, $\mathbf{J}_{\star}$ is the Jacobian of the corresponding isoparametric transformation and $\mathtt{n}^{\star}_{\mathtt{en}}$ is the number of elemental nodes of the approximation.

The following matrices and vector are thus defined as

$$\left[\mathbf{A}_{\star\star}\right]_e := \kappa \sum_{\mathtt{g}=1}^{\mathtt{n}^{\star}_{\mathtt{ip}}} \boldsymbol{\mathcal{Q}}^{\star}(\xi^{\star}_{\mathtt{g}}) \boldsymbol{\mathcal{Q}}^{\star T}(\xi^{\star}_{\mathtt{g}}) |\mathbf{J}_{\star}(\xi^{\star}_{\mathtt{g}})| w^{\star}_{\mathtt{g}},$$

$$\left[\mathbf{a}_{\star\lambda}\right]_e := \sum_{\mathtt{g}=1}^{\mathtt{n}^{\star}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}^{\star}(\xi^{\star}_{\mathtt{g}}) |\mathbf{J}_{\star}(\xi^{\star}_{\mathtt{g}})| w^{\star}_{\mathtt{g}},$$

$$\left[\mathbf{A}_{\star q}\right]_e := - \sqrt{\kappa} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\star}_{\mathtt{ip}}} \boldsymbol{\mathcal{Q}}^{\star}(\xi^{\star}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{\star T}_{\mathtt{n}_{\mathtt{sd}}}(\xi^{\star}_{\mathtt{g}}) |\mathbf{J}_{\star}(\xi^{\star}_{\mathtt{g}})| w^{\star}_{\mathtt{g}},$$

where $\xi^{\star}_{\mathtt{g}}$ and $w^{\star}_{\mathtt{g}}$ are the $\mathtt{n}^{\star}_{\mathtt{ip}}$ integration points and the weights associated with the higher order approximation in the space $\mathcal{V}^h_{\star}$.

Appendix 2: Matrices and Vectors for the Stokes Solver

In this appendix, the expressions of the matrices and vectors appearing in the discrete form of the HDG local and global problems for the Stokes equations are presented. It is worth recalling that in this problem, the primal, $\mathbf{u}$, and hybrid, $\widehat{\mathbf{u}}$, variables are $\mathtt{n}_{\mathtt{sd}}$-dimensional vector-valued unknowns and the mixed variable $\mathbf{L}$ is a tensor-valued unknown of dimension $\mathtt{n}_{\mathtt{sd}} \times \mathtt{n}_{\mathtt{sd}}$.

In addition to the compact forms of the shape functions presented in equations (42) and (44), the following definitions are introduced for the vectorial problem

$$\boldsymbol{\mathcal{N}}_{\mathtt{m}_{\mathtt{sd}}} := \left[N_1\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \; N_2\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \; \dots \; N_{\mathtt{n}_{\mathtt{en}}}\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \right]^T, \tag{45a}$$

$$\boldsymbol{\mathcal{N}}^{\star}{}_{\mathtt{m}_{\mathtt{sd}}} := \left[N_1^{\star}\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \; N_2^{\star}\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \; \dots \; N^{\star}_{\mathtt{n}^{\star}_{\mathtt{en}}}\mathbf{I}_{\mathtt{m}_{\mathtt{sd}}} \right]^T, \tag{45b}$$

$$\hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}} := \left[\hat{N}_1\mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; \hat{N}_2\mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \; \dots \; \hat{N}_{\mathtt{n}_{\mathtt{fn}}}\mathbf{I}_{\mathtt{n}_{\mathtt{sd}}} \right]^T, \tag{45c}$$

$$\hat{\boldsymbol{\mathcal{N}}}_{E} := \left[\hat{N}_1\boldsymbol{E} \; \hat{N}_2\boldsymbol{E} \; \dots \; \hat{N}_{\mathtt{n}_{\mathtt{fn}}}\boldsymbol{E} \right]^T, \tag{45d}$$

$$\hat{\boldsymbol{\mathcal{N}}}_{D} := \left[\hat{N}_1\boldsymbol{D} \; \hat{N}_2\boldsymbol{D} \; \dots \; \hat{N}_{\mathtt{n}_{\mathtt{fn}}}\boldsymbol{D} \right]^T, \tag{45e}$$

where $\mathtt{m}_{\mathtt{sd}} := \mathtt{n}_{\mathtt{sd}}^2$. Moreover, for each component $j = 1, \dots, \mathtt{n}_{\mathtt{sd}}$, the compact forms

$$\boldsymbol{\mathcal{N}}_{n_j} := \left[N_1 n_j \; N_2 n_j \; \dots \; N_{\mathtt{n}_{\mathtt{en}}} n_j \right]^T, \tag{45f}$$

$$\boldsymbol{\mathcal{Q}}_j := \left[[\mathbf{J}^{-1}\boldsymbol{\nabla} N_1]_j \; [\mathbf{J}^{-1}\boldsymbol{\nabla} N_2]_j \; \dots \; [\mathbf{J}^{-1}\boldsymbol{\nabla} N_{\mathtt{n}_{\mathtt{en}}}]_j \right]^T, \tag{45g}$$

$$\boldsymbol{\mathcal{Q}}^{\star}{}_j := \left[[\mathbf{J}_{\star}^{-1}\boldsymbol{\nabla} N_1^{\star}]_j \; [\mathbf{J}_{\star}^{-1}\boldsymbol{\nabla} N_2^{\star}]_j \; \dots \; [\mathbf{J}_{\star}^{-1}\boldsymbol{\nabla} N^{\star}_{\mathtt{n}^{\star}_{\mathtt{en}}}]_j \right]^T, \tag{45h}$$

are defined, n_j being the j-th component of the outward unit normal vector $\boldsymbol{n}$ to the face.

The solution of the HDG local problem (27) involves the matrices and vectors

$$\left[\mathbf{A}_{LL}\right]_e := -\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{m}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{m}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}},$$

$$[\mathbf{A}_{Lu}]_e := \sqrt{\nu} \sum_{j=1}^{\mathtt{n}_{\mathtt{sd}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{Q}}_j(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}},$$

$$[\mathbf{A}_{uu}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}},$$

$$[\mathbf{A}_{pu}]_e := \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{Q}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}},$$

$$[\mathbf{a}_{pp}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}},$$

$$[\mathbf{A}_{L\hat{u}}]_e := \sqrt{\nu} \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{j=1}^{\mathtt{n}_{\mathtt{sd}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{n_j}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}),$$

$$[\mathbf{A}_{u\hat{u}}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}),$$

$$[\mathbf{A}_{p\hat{u}}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{n}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}),$$

$$[\mathbf{f}_{L}]_e := \sqrt{\nu} \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{j=1}^{\mathtt{n}_{\mathtt{sd}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{n_j}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{u}_{D}\left(\boldsymbol{x}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{D},$$

$$\begin{aligned}[\mathbf{f}_{u}]_e := &\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}}) \boldsymbol{s}\left(\boldsymbol{x}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}})\right) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{e}}_{\mathtt{g}})| w^{\mathtt{e}}_{\mathtt{g}} \\ &+ \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{u}_{D}\left(\boldsymbol{x}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{D},\end{aligned}$$

$$[\mathbf{f}_{p}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \boldsymbol{\mathcal{N}}_{n}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{u}_{D}\left(\boldsymbol{x}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{D}.$$

Similarly, the matrices and vectors for the global problem (29) are

$$\begin{aligned}{}_{e} := \sqrt{\nu} \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \Bigg\{ &- \left(\sum_{j=1}^{\mathtt{n}_{\mathtt{sd}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{E}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{n_j}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{S} \\ &+ \left(\sum_{j=1}^{\mathtt{n}_{\mathtt{sd}}} \sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{n_j}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}) (1 - \chi^{\mathtt{f}}_{S}) \Bigg\},\end{aligned}$$

$$\begin{aligned}[\mathbf{A}_{\hat{u}u}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \tau_{\mathtt{f}} \Bigg\{ &- \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{E}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{S} \\ &+ \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}) (1 - \chi^{\mathtt{f}}_{S}) \Bigg\},\end{aligned}$$

$$\begin{aligned}[\mathbf{A}_{\hat{u}p}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \Bigg\{ &- \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{E}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{n}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{S} \\ &+ \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{\mathcal{N}}^{T}_{n}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}) (1 - \chi^{\mathtt{f}}_{S}) \Bigg\},\end{aligned}$$

$$\begin{aligned}[\mathbf{A}_{\hat{u}\hat{u}}]_e := \sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \Bigg\{ &\left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{D}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{S} \\ &+ \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{E}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{S} \\ &- \tau_{\mathtt{f}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \hat{\boldsymbol{\mathcal{N}}}^{T}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) (1 - \chi^{\mathtt{f}}_{D}) (1 - \chi^{\mathtt{f}}_{S}) \Bigg\},\end{aligned}$$

$$[\mathbf{f}_{\hat{u}}]_e := -\sum_{\mathtt{f}=1}^{\mathtt{n}^{\mathtt{e}}_{\mathtt{fa}}} \left(\sum_{\mathtt{g}=1}^{\mathtt{n}^{\mathtt{f}}_{\mathtt{ip}}} \hat{\boldsymbol{\mathcal{N}}}_{\mathtt{n}_{\mathtt{sd}}}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}}) \boldsymbol{g}\left(\boldsymbol{x}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})\right) |\mathbf{J}(\boldsymbol{\xi}^{\mathtt{f}}_{\mathtt{g}})| w^{\mathtt{f}}_{\mathtt{g}} \right) \chi^{\mathtt{f}}_{N}.$$

In addition, if a problem with purely Dirichlet boundary conditions is solved, the constraint (20) gives rise to the vector

$$\left[\mathbf{a}_{\bar{\rho}\rho}\right]_e := \sum_{g=1}^{n_{ip}^e} \mathcal{N}(\xi_g^e)|\mathbf{J}(\xi_g^e)|w_g^e, \tag{46}$$

which is used to enforce the constraint using an appropriately defined Lagrange multiplier.

Finally, for the postprocess (32), the following matrices are required

$$\left[\mathbf{A}_{\star\star}\right]_e := \nu \sum_{j=1}^{n_{sd}} \sum_{g=1}^{n_{ip}^\star} \mathcal{Q}^\star{}_j(\xi_g^\star)\mathcal{Q}^{\star T}{}_j(\xi_g^\star)|\mathbf{J}_\star(\xi_g^\star)|w_g^\star,$$

$$[\mathbf{A}_{\star\lambda}]_e := \sum_{g=1}^{n_{ip}^\star} \mathcal{N}^\star{}_{n_{sd}}(\xi_g^\star)|\mathbf{J}_\star(\xi_g^\star)|w_g^\star,$$

$$[\mathbf{A}_{\star L}]_e := -\sqrt{\nu} \sum_{j=1}^{n_{sd}} \sum_{g=1}^{n_{ip}^\star} \mathcal{Q}^\star{}_j(\xi_g^\star)\mathcal{N}^{\star T}{}_{m_{sd}}(\xi_g^\star)|\mathbf{J}_\star(\xi_g^\star)|w_g^\star.$$

Appendix 3: An Interface with the Mesh Generator `Gmsh`

In this appendix, the interface between the high-order open-source mesh generator `Gmsh` [146] and `HDGlab` is described. Starting from the structure of the variable `mesh` introduced in Sect. 6.1, the algorithm to convert a mesh file from the *.msh* to the *.mat* format of `HDGlab` is presented (Fig. 48).

Remark 10 The routines described in this appendix are designed starting from the *.msh* ASCII file format version 2.2.

First, it is worth recalling that `HDGlab` offers the feature of utilising either equally-spaced or Fekete points for the approximation. `Gmsh` provides mesh discretisation based on equally-spaced nodal sets, whence the variable `optionNodes` is set to 0, see Sect. 6.1.

The function `convertMSHtoMAT` requires the definition of the following data concerning the mesh to be imported:

```
optionNodes = 0; % GMSH utilises uniform nodal distributions

%% Option setup
fileName = 'ringH1unifP2';
nsd = 2;          % Number of spatial dimensions
pDegree = 2;      % Polynomial degree
isPlotMesh = 1;   % Boolean variable to plot the imported mesh
outputPath = 'meshFiles';

%% Construct the reference element
[refElem, refFace] = getRefData(pDegree, nsd, optionNodes);
refElem = refElem(1, pDegree);
refFace = refFace(pDegree, pDegree);

%% Store the coordinates of the nodes with duplication
[X, T, faces, matElem] = scanMeshFileMSH(fileName, pDegree, nsd);

%% Extract internal and external faces
intFaces = getInternalFaces(T, refElem, refFace);
extFaces = getBoundaryFaces(T, faces, refElem, refFace);

%% Build the mat structure of the mesh
mesh = buildMeshStruct(X, T, matElem, intFaces, extFaces, refElem, optionNodes);

%% Save mesh structure in .mat file
meshFile = sprintf('%s/%s', outputPath, fileName);
save(meshFile, 'mesh');

%% Visualise the imported mesh
if isPlotMesh
    visualiseMesh
end
```

Fig. 48 Function `convertMSHtoMAT` to convert the mesh file from *.msh* to *.mat* format

- `fileName`: String that specifies the name of the *.msh* mesh file without extension. The mesh files are archived in the directory `example`.
- `nsd`: Scalar variable defining the number of spatial dimensions (either 2 or 3).
- `pDegree`: Scalar variable defining the degree of the polynomial order of the mesh.
- `isPlotMesh`: Boolean variable to activate the option for visualising the imported mesh with the nodal distribution.
- `outputPath`: String that specifies the location where the imported mesh will be stored. The default location is the directory `meshFiles`.

Given the number of spatial dimensions and the polynomial degree defined above, the `refElem` and `refFace` data structures are constructed.

```
%% Node permutation from GMSH to Matlab ...
    ordering
permNodes = nodeRenumberingMSH(pDegree, nsd);
T(:, 1:nOfElemNodes) = T(:, permNodes);
```

Fig. 49 Extract of the `scanMeshFileMSH` function that constructs the permutation vector for the appropriate renumbering of the mesh nodes

```
%% Store the list of elements containing ...
    each node
nodeToElem = zeros(nOfNodes, ...
    nOfMaxElemsOneNode);
indexNode = zeros(nOfNodes, 1);
for kElem = 1:nOfElements
    Te = T(kElem, :);
    for iLocalNode = 1:nOfElemNodes
        iGlobalNode = Te(iLocalNode);
        indexNode(iGlobalNode) = ...
            indexNode(iGlobalNode) + 1;
        nodeToElem(iGlobalNode, ...
            indexNode(iGlobalNode)) = kElem;
    end
end
```

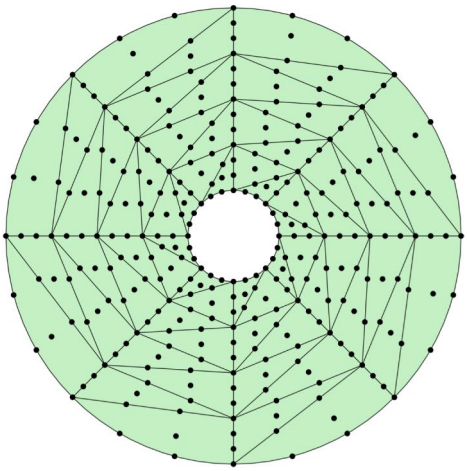
Fig. 51 Extract of the `getInternalFaces` function that constructs the list of elements containing each node

```
%% Construct the connectivity matrix
indexIni = 1;
for iElem = 1:mesh.nOfElements
    indexEnd = indexIni + refElem.nOfNodes ...
        - 1;
    mesh.indexT(iElem, 1) = indexIni;
    mesh.indexT(iElem, 2) = indexEnd;
    mesh.X(indexIni:indexEnd, :) = ...
        X(T(iElem,:), :);
    indexIni = indexEnd + 1;
end
mesh.nOfNodes = size(mesh.X,1);
```

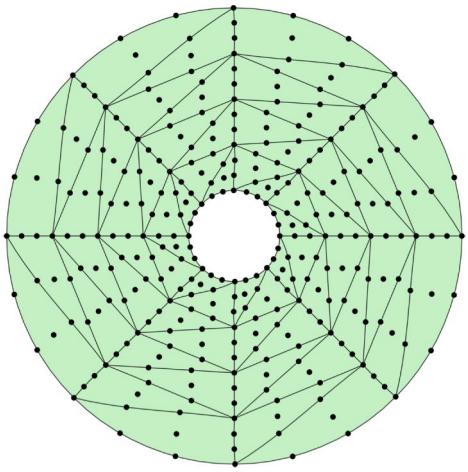
Fig. 52 Extract of the `buildMeshStruct` function that constructs the connectivity matrix

```
function [jElem, jFace, jPerm] = findNeighbourElement(T, iElem, faceNodes, ...
    listPotentialNeigh, elemFaces, nsd, nOfElemFaces, nOfFaceVertices)

sortFaceNodes = sort(faceNodes);

% Initialisation
jElem = 0;
jFace = 0;
jPerm = 0;

% Loop over potential neighbours
nOfPotentialNeigh = length(listPotentialNeigh(1, :));
for iPotentialNeigh = 1:nOfPotentialNeigh
    jIndexElem = listPotentialNeigh(1, iPotentialNeigh);
    if ~(jIndexElem == iElem)
        % Loop over faces of potential neighbour
        for jFace = 1:nOfElemFaces
            jFaceNodes = T(jIndexElem, elemFaces(jFace).nodes);
            % Compare set of nodes after sorting
            sortJFaceNode = sort(jFaceNodes);
            if sum(abs(sortFaceNodes-sortJFaceNode)) == 0
                jElem = jIndexElem;
                jPerm = getFacePermutation(jFaceNodes, faceNodes(1), nsd, nOfFaceVertices);
                return;
            end
        end
    end
end
```

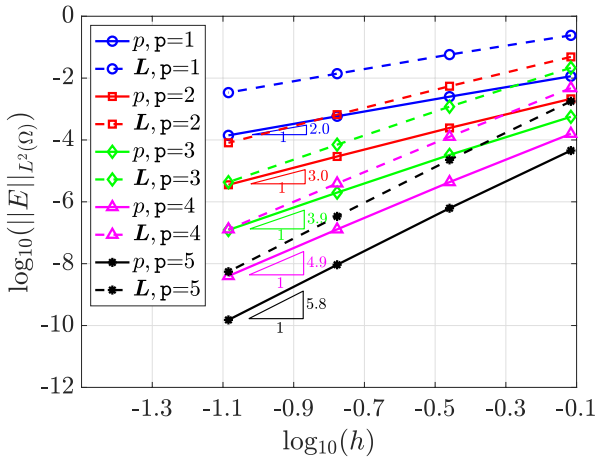
Fig. 50 Function `findNeighbourElement` that identifies the element sharing a face of a given element



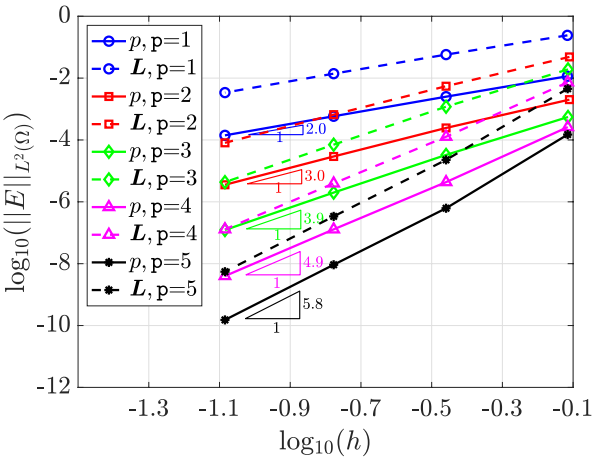
(a) Elasticity: nodal distribution



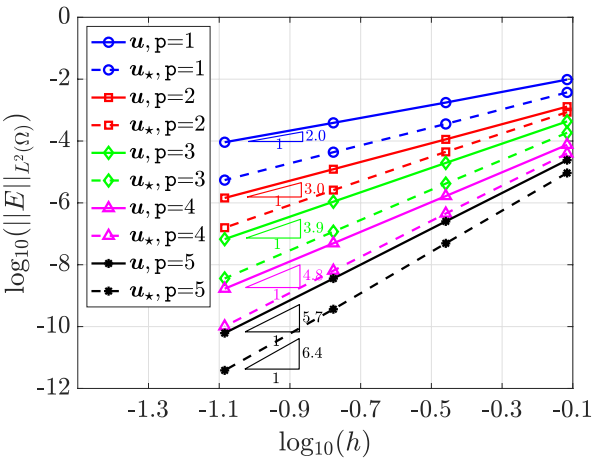
(b) Optimisation: nodal distribution



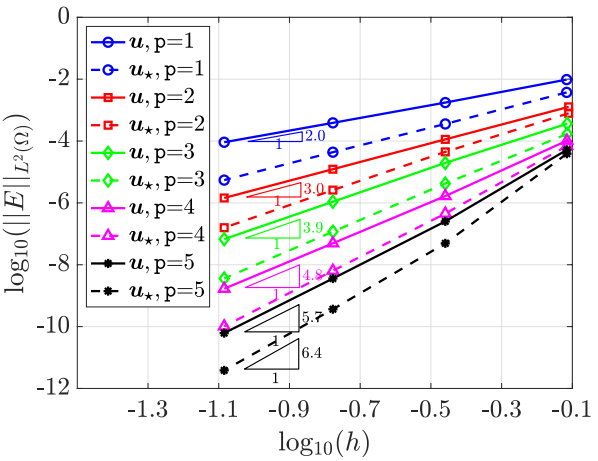
(c) Elasticity: h-convergence of p and $\boldsymbol{L}$



(d) Optimisation: h-convergence of p and $\boldsymbol{L}$



(e) Elasticity: h-convergence of $\boldsymbol{u}$ and $\boldsymbol{u}_\star$



(f) Optimisation: h-convergence of $\boldsymbol{u}$ and $\boldsymbol{u}_\star$

Fig. 53 Comparison of high-order meshes generated by `Gmsh` using the elastic approach (left) and the optimisation algorithm (right) for the Stokes equation in a ring domain. **a**, **b** Nodal distributions for meshes of degree 3. Convergence of the $\mathcal{L}_2(\Omega)$ error of **c**, **d** pressure, p, and mixed variable, $\boldsymbol{L}$, and **e**, **f** primal, $\boldsymbol{u}$, and postprocessed, $\boldsymbol{u}_\star$, velocities as a function of the characteristic mesh size h for polynomial degree of approxiomation $\mathtt{p} = 1, \dots, 5$

Reading the *.msh* File

The *.msh* file is read by the function `scanMeshFileMSH` and the following information is extracted:

- `X`: Array containing the coordinates of the mesh nodes.
- `T`: Connectivity matrix featuring the identifiers of the nodes associated with each element.
- `faces`: Array containing the information on the boundary faces, namely the identifier of the element the face belongs to, the list of vertices, the flags of the boundary and of the type of boundary condition imposed.
- `matElem`: Array containing the flag of the region each element belongs to.

A detailed description of the structure of the *.msh* file as well as the notation utilised by the mesh generator to identify element and face types is available in the official `Gmsh` documentation [13]. It is worth noting that the numbering of the nodes in `HDGlab` differs from the one provided by `Gmsh`. Hence, an appropriate permutation is performed as reported in Fig. 49.

Retrieving the Information on the Faces

As described in Sect. 6.1, the interior and exterior faces of the mesh are stored in the data structures `intFaces` and `extFaces`, respectively.

To construct the structure `intFaces`, the `getInternalFaces` function explores the mesh and, for each face of each element, it identifies the neighbouring element by comparing the list of face nodes, see Fig. 50. In order to optimise this operation, a list of potential neighbours is preliminarily stored by identifying the list of elements containing each node, as detailed by the extract in Fig. 51.

The function `getBoundaryFaces`, not reported here for brevity, is responsible for constructing the data structure of the exterior faces. More precisely, the structure `extFaces` is obtained by rearranging the information previously stored in the data structure `faces`, according to the rationale described in Sect. 6.1.

Assemblying the Mesh Structure

The structure `mesh` is finally assembled by the `buildMeshStruct` function using the information obtained by the mesh generator, namely `X`, `T` and `matElem`, and the structures of the interior and exterior faces, `intFaces` and `extFaces`, previously generated. More precisely, the connectivity matrix is constructed via a loop on the mesh elements as reported in Fig. 52.

It is worth noting that the framework provided in `HDGlab` to import meshes generated using `Gmsh` can be easily extended to any mesh generator by introducing appropriate functions to construct the `intFaces` and `extFaces` data structures, as described above.

Some Examples of High-Order Meshes Using `Gmsh`

In this section, several meshes of the ring domain introduced in Sect. 11.2 are generated using `Gmsh` and tested to verify the optimal convergence rate of the code when high-order meshes with equally-spaced nodal sets are considered. More precisely, Fig. 53 displays the comparison of the third order meshes provided by `Gmsh` using the `Mesh.HighOrderOptimize` option either with an elastic approach or an optimisation algorithm [13]. It is worth noting that the elastic approach mainly induces the curvature of the boundary of the domain, whereas the optimisation strategy is responsible for curved edges to appear in the interior of the domain as well (Fig. 53a, b). In both cases, the nodes are equally-spaced.

In addition, the $\mathcal{L}_2(\Omega)$ error of the pressure and the gradient of velocity (Fig 53c, d) and the primal and postprocessed velocities (Fig 53e, f) is reported as a function of the characteristic mesh size. The results diplay the expected optimal convergence of order $p+1$ for pressure, velocity and gradient of velocity and the superconvergence of the postprocessed variable $\boldsymbol{u}_\star$, showing the capability of `HDGlab` to work using both equally-spaced and Fekete nodal distributions. It is worth noting that the results obtained using Fekete nodal sets (Fig. 37) provide up to one order of magnitude of extra accuracy for a given mesh size compared to the equally-spaced nodes in Fig. 53.

References

1. `code\_aster`: structures and thermomechanics analysis for studies and research. https://code-aster.org/
2. `Code\_Saturne`: EDF open-source software to solve computational fluid dynamics applications. https://www.code-saturne.org/cms/
3. `DiSk++`: a C++ template library for discontinuous skeletal methods. https://github.com/wareHHOuse/diskpp
4. `Feel++`: a powerful, scalable and expressive finite element embedded library in C++. http://www.feelpp.org
5. `FESTUNG`: a MATLAB/GNU Octave toolbox for the discontinuous Galerkin method. https://github.com/FESTUNG/FESTUNG
6. `Firedrake`: an automated system for the solution of partial differential equations using the finite element method. https://www.firedrakeproject.org
7. `GetFEM`: an open-source finite element library. http://getfem.org
8. `HArDCore`: Hybrid Arbitrary Degree::Core. https://github.com/jdroniou/HArDCore

9. `HDG3D`: Matlab implementation of the hybridizable discontinuous Galerkin method on general tetrahedrizations of polyhedra in three dimensional space. https://github.com/team-pancho/HDG3D
10. `Nektar++`: spectral/hp element framework. https://www.nektar.info
11. `Netgen/NGSolve`: a high performance multiphysics finite element software. https://ngsolve.org
12. `deal.II`: an open source finite element library. https://www.dealii.org
13. `Gmsh`: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. https://gmsh.info
14. `MFEM`: modular finite element methods library. https://mfem.org
15. Abbas M, Ern A, Pignet N (2018) Hybrid high-order methods for finite deformations of hyperelastic materials. Comput Mech 62(4):909–928
16. Abbas M, Ern A, Pignet N (2019) A hybrid high-order method for finite elastoplastic deformations within a logarithmic strain framework. Int J Numer Methods Eng 120(3):303–327
17. Abbas M, Ern A, Pignet N (2019) A hybrid high-order method for incremental associative plasticity with small deformations. Comput Methods Appl Mech Eng 346:891–912
18. Agullo E, Giraud L, Gobé A, Kuhn M, Lanteri S, Moya L (2020) High order HDG method and domain decomposition solvers for frequency-domain electromagnetics. Int J Numer Model Electron Netw Dev Fields 33(2):e2678
19. Ainsworth M, Fu G (2018) Fully computable a posteriori error bounds for hybridizable discontinuous Galerkin finite element approximations. J Sci Comput 77(1):443–466
20. Anderson R, Andrej J, Barker A, Bramwell J, Camier JS, Cerveny J, Dobrev V, Dudouit Y, Fisher A, Kolev T, Pazner W, Stowell M, Tomov V, Dahm J, Medina D, Zampini S (2020) MFEM: a modular finite element methods library. Technical report, arXiv arXiv:1911.09220
21. Anderson TH, Civiletti BJ, Monk PB, Lakhtakia A (2020) Coupled optoelectronic simulation and optimization of thin-film photovoltaic solar cells. J Comput Phys 407:109,242
22. Araya R, Solano M, Vega P (2019) Analysis of an adaptive HDG method for the Brinkman problem. IMA J Numer Anal 39(3):1502–1528
23. Araya R, Solano M, Vega P (2019) A posteriori error analysis of an HDG method for the Oseen problem. Appl Numer Math 146:291–308
24. Arbogast T, Pencheva G, Wheeler MF, Yotov I (2007) A multiscale mortar mixed finite element method. Multiscale Model Simul 6(1):319–346
25. Bangerth W, Hartmann R, Kanschat G (2007) deal.II—a general purpose object oriented finite element library. ACM Trans Math Softw 33(4):24/1–24/27
26. Barrenechea GR, Bosy M, Dolean V, Nataf F, Tournier PH (2018) Hybrid discontinuous Galerkin discretisation and domain decomposition preconditioners for the Stokes problem. Comput Methods Appl Math. https://doi.org/10.1515/cmam-2018-0005
27. Bonaldi F, Di Pietro DA, Geymonat G, Krasucki F (2018) A hybrid high-order method for Kirchhoff–Love plate bending problems. ESAIM Math Model Numer Anal 52(2):393–421
28. Bonnasse-Gahot M, Calandra H, Diaz J, Lanteri S (2017) Hybridizable discontinuous Galerkin method for the 2-D frequency-domain elastic wave equations. Geophys J Int 213(1):637–659
29. Botti L, Di Pietro DA (2018) Assessment of hybrid high-order methods on curved meshes and comparison with discontinuous Galerkin methods. J Comput Phys 370:58–84
30. Botti L, Di Pietro DA, Droniou J (2018) A hybrid high-order discretisation of the Brinkman problem robust in the Darcy and Stokes limits. Comput Methods Appl Mech Eng 341:278–310
31. Botti L, Di Pietro DA, Droniou J (2019) A hybrid high-order method for the incompressible Navier–Stokes equations based on Temam's device. J Comput Phys 376:786–816
32. Botti M, Di Pietro DA, Guglielmana A (2019) A low-order nonconforming method for linear elasticity on general meshes. Comput Methods Appl Mech Eng 354:96–118
33. Botti M, Di Pietro DA, Le Maître O, Sochala P (2020) Numerical approximation of poroelasticity with random coefficients using polynomial chaos and hybrid high-order methods. Comput Methods Appl Mech Eng 361:112,736
34. Botti M, Di Pietro DA, Sochala P (2017) A hybrid high-order method for nonlinear elasticity. SIAM J Numer Anal 55(6):2687–2717
35. Botti M, Di Pietro DA, Sochala P (2020) A hybrid high-order discretization method for nonlinear poroelasticity. Comput Methods Appl Math 20(2):227–249
36. Brezzi F, Fortin M (1991) Mixed and hybrid finite elements methods. Springer series in computational mathematics. Springer, Berlin
37. Bui-Thanh T (2015) From Godunov to a unified hybridized discontinuous Galerkin framework for partial differential equations. J Comput Phys 295:114–146
38. Bui-Thanh T (2016) Construction and analysis of HDG methods for linearized shallow water equations. SIAM J Sci Comput 38(6):A3696–A3719
39. Burman E, Delay G, Ern A (2020) An unfitted hybrid high-order method for the Stokes interface problem. IMA J Numer Anal. https://doi.org/10.1093/imanum/draa059
40. Burman E, Ern A (2018) An unfitted hybrid high-order method for elliptic interface problems. SIAM J Numer Anal 56(3):1525–1546
41. Camargo L, López-Rodríguez B, Osorio M, Solano M (2020) An HDG method for Maxwell's equations in heterogeneous media. Comput Methods Appl Mech Eng 368:113178
42. Cangiani A, Dong Z, Georgoulis E, Houston P (2017) *hp*-version discontinuous Galerkin methods on polygonal and polyhedral meshes. Springer, Berlin
43. Cantwell CD, Moxey D, Comerford A, Bolis A, Rocco G, Mengaldo G, De Grazia D, Yakovlev S, Lombard JE, Ekelschot D, Jordi B, Xu H, Mohamied Y, Eskilsson C, Nelson B, Vos P, Biotto C, Kirby RM, Sherwin SJ (2015) Nektar++: an open-source spectral/hp element framework. Comput Phys Commun 192:205–219
44. Cascavita KL, Bleyer J, Chateau X, Ern A (2018) Hybrid discretization methods with adaptive yield surface detection for Bingham pipe flows. J Sci Comput 77(3):1424–1443
45. Castanon Quiroz D, Di Pietro DA (2020) a hybrid high-order method for the incompressible Navier–Stokes problem robust for large irrotational body forces. Comput Math Appl 79(9):2655–2677
46. Castillo P, Gómez S (2020) Conservative super-convergent and hybrid discontinuous Galerkin methods applied to nonlinear Schrödinger equations. Appl Math Comput 371:124,950
47. Celiker F, Cockburn B, Shi K (2010) Hybridizable discontinuous Galerkin methods for Timoshenko beams. J Sci Comput 44(1):1–37
48. Celiker F, Cockburn B, Shi K (2011) A projection-based error analysis of HDG methods for Timoshenko beams. Math Comput 81(277):131–151
49. Cesmelioglu A, Cockburn B, Nguyen NC, Peraire J (2013) Analysis of HDG methods for Oseen equations. J Sci Comput 55(2):392–431
50. Cesmelioglu A, Cockburn B, Qiu W (2017) Analysis of a hybridizable discontinuous Galerkin method for the steady-state incompressible Navier–Stokes equations. Math Comput 86(306):1643–1670

51. Cesmelioglu A, Rhebergen S, Wells GN (2020) An embedded-hybridized discontinuous Galerkin method for the coupled Stokes–Darcy system. J Comput Appl Math 367:112,476
52. Chave F, Di Pietro DA, Formaggia L (2019) A hybrid high-order method for passive transport in fractured porous media. GEM Int J Geomath 10(1):12
53. Chave F, Di Pietro DA, Marche F, Pigeonneaux F (2016) A hybrid high-order method for the Cahn–Hilliard problem in mixed form. SIAM J Numer Anal 54(3):1873–1898
54. Chen G, Cockburn B, Singler J, Zhang Y (2019) Superconvergent interpolatory HDG methods for reaction diffusion equations I: an HDG_k method. J Sci Comput 81(3):2188–2212
55. Chen G, Cui J, Xu L (2019) Analysis of a hybridizable discontinuous Galerkin method for the Maxwell operator. ESAIM Math Model Numer Anal 53(1):301–324
56. Chen G, Monk P, Zhang Y (2019) An HDG method for the time-dependent drift-diffusion model of semiconductor devices. J Sci Comput 80:420–443
57. Chen H, Li J, Qiu W (2014) Robust a posteriori error estimates for HDG method for convection–diffusion equations. IMA J Numer Anal 36(1):437–462
58. Chen H, Qiu W, Shi K (2018) A priori and computable a posteriori error estimates for an HDG method for the coercive Maxwell equations. Comput Methods Appl Mech Eng 333:287–310
59. Chen H, Qiu W, Shi K, Solano M (2017) A superconvergent HDG Method for the Maxwell equations. J Sci Comput 70(3):1010–1029
60. Chen Y, Cockburn B (2012) Analysis of variable-degree HDG methods for convection–diffusion equations. Part I: general nonconforming meshes. IMA J Numer Anal 32(4):1267–1293
61. Chen Y, Cockburn B (2014) Analysis of variable-degree HDG methods for convection–diffusion equations. Part II: semimatching nonconforming meshes. Math Comput 83(285):87–111
62. Chen Y, Cockburn B, Dong B (2016) Superconvergent HDG methods for linear, stationary, third-order equations in one-space dimension. Math Comput 85(302):2715–2742
63. Chen Y, Dong B, Jiang J (2018) Optimally convergent hybridizable discontinuous Galerkin method for fifth-order Korteweg–de Vries type equations. ESAIM Math Model Numer Anal 52(6):2283–2306
64. Childs PR (2010) Rotating flow. Elsevier, Amsterdam
65. Cho K, Moon M (2020) Multiscale hybridizable discontinuous Galerkin method for elliptic problems in perforated domains. J Comput Appl Math 365:112,346
66. Chouly F, Ern A, Pignet N (2020) A hybrid high-order discretization combined with Nitsche's method for contact and Tresca friction in small strain elasticity. SIAM J Sci Comput 42(4):A2300–A2324
67. Christophe A, Descombes S, Lanteri S (2018) An implicit hybridized discontinuous Galerkin method for the 3D time-domain Maxwell equations. Appl Math Comput 319:395–408
68. Chung E, Cockburn B, Fu G (2014) The staggered DG method is the limit of a hybridizable DG method. SIAM J Numer Anal 52(2):915–932
69. Chung E, Cockburn B, Fu G (2016) The staggered DG method is the limit of a hybridizable DG method. Part II: the Stokes flow. J Sci Comput 66(2):870–887
70. Chung E, Efendiev Y, Leung WT (2019) Generalized multiscale finite element methods with energy minimizing oversampling. Int J Numer Methods Eng 117(3):316–343
71. Cicuttin M, Di Pietro D, Ern A (2018) Implementation of discontinuous skeletal methods on arbitrary-dimensional, polytopal meshes using generic programming. J Comput Appl Math 344:852–874
72. Cicuttin M, Ern A, Gudi T (2020) Hybrid high-order methods for the elliptic obstacle problem. J Sci Comput 83(1):8
73. Cicuttin M, Ern A, Lemaire S (2018) A hybrid high-order method for highly oscillatory elliptic problems. Comput Methods Appl Math 19(4):723–748
74. Ciucă C, Fernandez P, Christophe A, Nguyen NC, Peraire J (2020) Implicit hybridized discontinuous Galerkin methods for compressible magnetohydrodynamics. J Comput Phys X 5:100,042
75. Cockburn B (2016) Static condensation, hybridization, and the devising of the HDG methods. In: Barrenechea GR, Brezzi F, Cangiani A, Georgoulis E (eds) Building bridges: connections and challenges in modern approaches to numerical partial differential equations. Springer, Cham, pp 129–177
76. Cockburn B, Mustapha K (2015) A hybridizable discontinuous Galerkin method for fractional diffusion problems. Numer Math 130(2):293–314
77. Cockburn B, Dong B, Guzmán J (2008) A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. Math Comput 77(264):1887–1916
78. Cockburn B, Dong B, Guzmán J (2009) A hybridizable and superconvergent discontinuous Galerkin method for biharmonic problems. J Sci Comput 40(1–3):141–187
79. Cockburn B, Dong B, Guzmán J, Restelli M, Sacco R (2009) A hybridizable discontinuous Galerkin method for steady-state convection–diffusion–reaction problems. SIAM J Sci Comput 31(5):3827–3846
80. Cockburn B, Dubois O, Gopalakrishnan J, Tan S (2014) Multigrid for an HDG method. IMA J Numer Anal 34(4):1386–1425
81. Cockburn B, Fu G (2017) Devising superconvergent HDG methods with symmetric approximate stresses for linear elasticity by M-decompositions. IMA J Numer Anal 38(2):566–604
82. Cockburn B, Fu G (2017) Superconvergence by M-decompositions. Part II: construction of two-dimensional finite elements. ESAIM Math Model Numer Anal 51(1):165–186
83. Cockburn B, Fu G (2017) Superconvergence by M-decompositions. Part III: construction of three-dimensional finite elements. ESAIM Math Model Numer Anal 51(1):365–398
84. Cockburn B, Fu G, Qiu W (2017) A note on the devising of superconvergent HDG methods for Stokes flow by M-decompositions. IMA J Numer Anal 37(2):730–749
85. Cockburn B, Fu G, Sayas FJ (2017) Superconvergence by M -decompositions. Part I: general theory for HDG methods for diffusion. Math Comput 86(306):1609–1641
86. Cockburn B, Fu Z, Hungria A, Ji L, Sánchez MA, Sayas FJ (2018) Stormer–Numerov HDG methods for acoustic waves. J Sci Comput 75(2):597–624
87. Cockburn B, Gopalakrishnan J (2004) A characterization of hybridized mixed methods for second order elliptic problems. SIAM J Numer Anal 42(1):283–301
88. Cockburn B, Gopalakrishnan J (2009) The derivation of hybridizable discontinuous Galerkin methods for Stokes flow. SIAM J Numer Anal 47(2):1092–1125
89. Cockburn B, Gopalakrishnan J, Lazarov R (2009) Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems. SIAM J Numer Anal 47(2):1319–1365
90. Cockburn B, Gopalakrishnan J, Nguyen NC, Peraire J, Sayas FJ (2011) Analysis of HDG methods for Stokes flow. Math Comput 80(274):723–760
91. Cockburn B, Karniadakis GE, Shu CW (2000) The development of discontinuous Galerkin methods. In: Discontinuous Galerkin methods (Newport, RI, 1999), lecture notes computer science engineering, vol 11. Springer, Berlin, pp 3–50
92. Cockburn B, Nguyen NC, Peraire J (2010) A comparison of HDG methods for Stokes flow. J Sci Comput 45(1–3):215–237

93. Cockburn B, Quenneville-Bélair V (2014) Uniform-in-time superconvergence of the HDG methods for the acoustic wave equation. Math Comput 83(285):65–85
94. Cockburn B, Sayas FJ (2014) Divergence-conforming HDG methods for Stokes flows. Math Comput 83(288):1571–1598
95. Cockburn B, Shen J (2016) A hybridizable discontinuous Galerkin method for the *p*-Laplacian. SIAM J Sci Comput 38(1):A545–A566
96. Cockburn B, Shen J (2019) An algorithm for stabilizing hybridizable discontinuous Galerkin methods for nonlinear elasticity. Res Appl Math 1:100001
97. Cockburn B, Shi K (2013) Superconvergent HDG methods for linear elasticity with weakly symmetric stresses. IMA J Numer Anal 33(3):747–770
98. Cockburn B, Shi K (2014) Devising HDG methods for Stokes flow: an overview. Comput Fluids 98:221–229
99. Cockburn B, Shu CW (1998) The local discontinuous Galerkin method for time-dependent convection–diffusion systems. SIAM J Numer Anal 35(6):2440–2463
100. Cockburn B, Singler JR, Zhang Y (2019) Interpolatory HDG method for parabolic semilinear PDEs. J Sci Comput 79(3):1777–1800
101. Cockburn B, Solano M (2012) Solving Dirichlet boundary-value problems on curved domains by extensions from subdomains. SIAM J Sci Comput 34(1):A497–A519
102. Cockburn B, Solano M (2014) Solving convection–diffusion problems on curved domains by extensions from subdomains. J Sci Comput 59(2):512–543
103. Cockburn B, Wang Z (2017) Adjoint-based, superconvergent Galerkin approximations of linear functionals. J Sci Comput 73(2–3):644–666
104. Cockburn B, Zhang W (2012) A posteriori error estimates for HDG methods. J Sci Comput 51(3):582–607
105. Cockburn B, Zhang W (2013) A posteriori error analysis for hybridizable discontinuous Galerkin methods for second order elliptic problems. SIAM J Numer Anal 51(1):676–693
106. Cockburn B, Di Pietro DA, Ern A (2016) Bridging the hybrid high-order and hybridizable discontinuous Galerkin methods. ESAIM Math Model Numer Anal 50(3):635–650
107. Costa-Solé A, Ruiz-Gironés E, Sarrate J (2019) An HDG formulation for incompressible and immiscible two-phase porous media flow problems. Int J Comput Fluid Dyn 33(4):137–148
108. Di Pietro D, Ern A (2012) Mathematical aspects of discontinuous Galerkin methods, vol 69. Springer, Heidelberg
109. Di Pietro D, Ern A (2015) A hybrid high-order locking-free method for linear elasticity on general meshes. Comput Methods Appl Mech Eng 283:1–21
110. Di Pietro D, Ern A, Lemaire S (2014) An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators. Comput Methods Appl Math 14(4):461–472
111. Di Pietro DA, Droniou J (2017) A hybrid high-order method for Leray–Lions elliptic equations on general meshes. Math Comput 86(307):2159–2191
112. Di Pietro DA, Droniou J (2020) The hybrid high-order method for polytopal meshes. Modeling, simulation and applications series. Springer, Berlin
113. Di Pietro DA, Droniou J, Ern A (2015) A discontinuous-skeletal method for advection–diffusion–reaction on general meshes. SIAM J Numer Anal 53(5):2135–2157
114. Di Pietro DA, Droniou J, Manzini G (2018) Discontinuous skeletal gradient discretisation methods on polytopal meshes. J Comput Phys 355:397–425
115. Di Pietro DA, Ern A (2015) Hybrid high-order methods for variable-diffusion problems on general meshes. C R Math 353(1):31–34
116. Di Pietro DA, Ern A, Linke A, Schieweck F (2016) A discontinuous skeletal method for the viscosity-dependent Stokes problem. Comput Methods Appl Mech Eng 306:175–195
117. Di Pietro DA, Krell S (2018) A hybrid high-order method for the steady incompressible Navier–Stokes problem. J Sci Comput 74(3):1677–1705
118. Diskin B, Thomas JL (2011) Comparison of node-centered and cell-centered unstructured finite-volume discretizations: inviscid fluxes. AIAA J 49(4):836–854
119. Diskin B, Thomas JL, Nielsen EJ, Nishikawa H, White JA (2010) Comparison of node-centered and cell-centered unstructured finite-volume discretizations: viscous fluxes. AIAA J 48(7):1326
120. Donea J, Huerta A (2003) Finite element methods for flow problems. Wiley, Hoboken
121. Dong H, Wang B, Xie Z, Wang LL (2016) An unfitted hybridizable discontinuous Galerkin method for the Poisson interface problem and its error analysis. IMA J Numer Anal 37(1):444–476
122. Du S, Sayas FJ (2020) A unified error analysis of hybridizable discontinuous Galerkin methods for the static Maxwell equations. SIAM J Numer Anal 58(2):1367–1391
123. Efendiev Y, Lazarov R, Moon M, Shi K (2015) A spectral multiscale hybridizable discontinuous Galerkin method for second order elliptic problems. Comput Methods Appl Mech Eng 292:243–256 Special Issue on Advances in Simulations of Subsurface Flow and Transport (Honoring Professor Mary F. Wheeler)
124. Egger H, Schöberl J (2009) A hybrid mixed discontinuous Galerkin finite-element method for convection–diffusion problems. IMA J Numer Anal 30(4):1206–1234
125. Egger H, Waluga C (2012) *hp*-analysis of a hybrid DG method for Stokes flow. IMA J Numer Anal 33(2):687–721
126. Egger H, Waluga C (2012) A hybrid mortar method for incompressible flow. Int J Numer Anal Model 9(4):793–812
127. Fabien MS (2020) A GPU-accelerated hybridizable discontinuous Galerkin method for linear elasticity. Commun Comput Phys 27(2):513–545
128. Fabien MS (2020) A high-order implicit HDG method for the Benjamin–Bona–Mahony equation. Int J Numer Methods Fluids. https://doi.org/10.1002/fld.4896
129. Fabien MS, Knepley MG, Mills RT, Riviere BM (2019) Manycore parallel computing for a hybridizable discontinuous Galerkin nested multigrid method. SIAM J Sci Comput 41(2):C73–C96
130. Fabien MS, Knepley MG, Rivière BM (2018) A hybridizable discontinuous Galerkin method for two-phase flow in heterogeneous porous media. Int J Numer Methods Eng 116(3):161–177
131. Farahinia A, Zhang WJ (2019) Numerical investigation into the mixing performance of micro T-mixers with different patterns of obstacles. J Braz Soc Mech Sci Eng 41(11):491
132. Fernandez P, Christophe A, Terrana S, Nguyen NC, Peraire J (2018) Hybridized discontinuous Galerkin methods for wave propagation. J Sci Comput 77(3):1566–1604
133. Fernandez P, Nguyen NC, Peraire J (2017) The hybridized discontinuous Galerkin method for implicit large-eddy simulation of transitional turbulent flows. J Comput Phys 336:308–329
134. Fidkowski KJ (2016) A hybridized discontinuous Galerkin method on mapped deforming domains. Comput Fluids 139:80–91
135. Fidkowski KJ (2019) Comparison of hybrid and standard discontinuous Galerkin methods in a mesh-optimisation setting. Int J Comput Fluid Dyn 33(1–2):34–42
136. Fidkowski KJ, Chen G (2020) Output-based mesh optimization for hybridized and embedded discontinuous Galerkin methods. Int J Numer Methods Eng 121(5):867–887

137. Fournier Y, Bonelle J, Moulinec C, Shang Z, Sunderland AG, Uribe JC (2011) Optimizing Code_Saturne computations on Petascale systems. Comput Fluids 45(1):103–108
138. Fraeijs de Veubeke B (1965) Displacement and equilibrium models in the finite element method. In: Zienkiewicz OC, Holister GS (eds) Stress analysis. Wiley, Hoboken, pp 145–197
139. Franciolini M, Fidkowski KJ, Crivellini A (2020) Efficient discontinuous Galerkin implementations and preconditioners for implicit unsteady compressible flow simulations. Comput Fluids 203:104542
140. Fu G (2020) Arbitrary Lagrangian–Eulerian hybridizable discontinuous Galerkin methods for incompressible flow with moving boundaries and interfaces. Comput Methods Appl Mech Eng 367:113158
141. Fu G, Cockburn B, Stolarski H (2015) Analysis of an HDG method for linear elasticity. Int J Numer Methods Eng 102(3–4):551–575
142. Fu G, Jin Y, Qiu W (2018) Parameter-free superconvergent H(div)-conforming HDG methods for the Brinkman equations. IMA J Numer Anal 39(2):957–982
143. Fu Z, Gatica LF, Sayas FJ (2015) Algorithm 949: MATLAB tools for HDG in three dimensions. ACM Trans Math Softw 41(3):1–21
144. Gander MJ, Hajian S (2018) Analysis of Schwarz methods for a hybridizable discontinuous Galerkin discretization: the many-subdomain case. Math Comput 87(312):1635–1657
145. Gatica GN, Sequeira FA (2015) Analysis of an augmented HDG method for a class of quasi-Newtonian Stokes flows. J Sci Comput 65(3):1270–1308
146. Geuzaine C, Remacle JF (2009) Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 79(11):1309–1331
147. Giacomini M, Borchini L, Sevilla R, Huerta A (2020) Separated response surfaces for flows in parametrised domains: comparison of a priori and a posteriori PGD algorithms. Technical report, arXiv arXiv:2009.02176. Submitted
148. Giacomini M, Karkoulias A, Sevilla R, Huerta A (2018) A superconvergent HDG method for Stokes flow with strongly enforced symmetry of the stress tensor. J Sci Comput 77(3):1679–1702
149. Giacomini M, Sevilla R (2019) Discontinuous Galerkin approximations in computational mechanics: hybridization, exact geometry and degree adaptivity. SN Appl Sci 1:1047
150. Giacomini M, Sevilla R (2020) A second-order face-centred finite volume method on general meshes with automatic mesh adaptation. Int J Numer Methods Eng. https://doi.org/10.1002/nme.6428
151. Giacomini M, Sevilla R, Huerta A (2020) Tutorial on hybridizable discontinuous Galerkin (HDG) formulation for incompressible flow problems. In: Lorenzis LD, Düster A (eds) Modeling in engineering using innovative numerical methods for solids and fluids. CISM International Centre for Mechanical Sciences, vol 599. Springer, Berlin, pp 163–201
152. Giorgiani G, Fernández-Méndez S, Huerta A (2013) Hybridizable discontinuous Galerkin p-adaptivity for wave propagation problems. Int J Numer Methods Fluids 72(12):1244–1262
153. Giorgiani G, Fernández-Méndez S, Huerta A (2014) Hybridizable discontinuous Galerkin with degree adaptivity for the incompressible Navier–Stokes equations. Comput Fluids 98:196–208
154. Gürkan C, Kronbichler M, Fernández-Méndez S (2017) Extended hybridizable discontinuous Galerkin with Heaviside enrichment for heat bimaterial problems. J Sci Comput 72(2):542–567
155. Gürkan C, Kronbichler M, Fernández-Méndez S (2019) Extended hybridizable discontinuous Galerkin for incompressible flow problems with unfitted meshes and interfaces. Int J Numer Methods Eng 117(7):756–777
156. Gürkan C, Sala-Lardies E, Kronbichler M, Fernández-Méndez S (2016) eXtended Hybridizable Discontinous Galerkin (X-HDG) for void problems. J Sci Comput 66(3):1313–1333
157. Guyan R (1965) Reduction of stiffness and mass matrices. AIAA J 3(2):380
158. Hesthaven J, Warburton T (2002) Nodal high-order methods on unstructured grids: I. Time-domain solution of Maxwell's equations. J Comput Phys 181(1):186–221
159. Hoermann JM, Bertoglio C, Kronbichler M, Pfaller MR, Chabiniok R, Wall WA (2018) An adaptive hybridizable discontinuous Galerkin approach for cardiac electrophysiology. Int J Numer Methods Biomed Eng 34(5):e2959
160. Horváth TL, Rhebergen S (2019) A locally conservative and energy-stable finite-element method for the Navier–Stokes problem on time-dependent domains. Int J Numer Methods Fluids 89(12):519–532
161. Horváth TL, Rhebergen S (2020) An exactly mass conserving space-time embedded-hybridized discontinuous Galerkin method for the Navier–Stokes equations on moving domains. J Comput Phys 417:109,577
162. Huang J, Huang X (2019) A hybridizable discontinuous Galerkin method for Kirchhoff plates. J Sci Comput 78(1):290–320
163. Huerta A, Angeloski A, Roca X, Peraire J (2013) Efficiency of high-order elements for continuous and discontinuous Galerkin methods. Int J Numer Methods Eng 96(9):529–560
164. Hungria A, Prada D, Sayas FJ (2017) HDG methods for elastodynamics. Comput Math Appl 74(11):2671–2690
165. Huynh LNT, Nguyen NC, Peraire J, Khoo BC (2013) A high-order hybridizable discontinuous Galerkin method for elliptic interface problems. Int J Numer Methods Eng 93(2):183–200
166. Jaust A, Reuter B, Aizinger V, Schütz J, Knabner P (2018) FESTUNG: a MATLAB/GNU Octave toolbox for the discontinuous Galerkin method. Part III: hybridized discontinuous Galerkin (HDG) formulation. Comput Math Appl 75(12):4505–4533
167. Kabaria H, Lew AJ, Cockburn B (2015) A hybridizable discontinuous Galerkin formulation for non-linear elasticity. Comput Methods Appl Mech Eng 283:303–329
168. Kang S, Bui-Thanh T, Arbogast T (2019) A hybridized discontinuous Galerkin method for a linear degenerate elliptic equation arising from two-phase mixtures. Comput Methods Appl Mech Eng 350:315–336
169. Kang S, Giraldo FX, Bui-Thanh T (2020) IMEX HDG-DG: a coupled implicit hybridized discontinuous Galerkin and explicit discontinuous Galerkin approach for shallow water systems. J Comput Phys 401:109010
170. Kirby R, Sherwin SJ, Cockburn B (2011) To CG or to HDG: a comparative study. J Sci Comput 51(1):183–212
171. Kirk KLA, Rhebergen S (2019) Analysis of a pressure-robust hybridized discontinuous Galerkin method for the stationary Navier–Stokes equations. J Sci Comput 81(2):881–897
172. Komala-Sheshachala S, Sevilla R, Hassan O (2020) A coupled HDG-FV scheme for the simulation of transient inviscid compressible flows. Comput Fluids 202:104495
173. Kronbichler M, Schoeder S, Müller C, Wall WA (2016) Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. Int J Numer Methods Eng 106(9):712–739
174. Kronbichler M, Wall WA (2018) A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. SIAM J Sci Comput 40(5):A3423–A3448
175. La Spina A, Giacomini M, Huerta A (2020) Hybrid coupling of CG and HDG discretizations based on Nitsche's method. Comput Mech 65(2):311–330
176. La Spina A, Kronbichler M, Giacomini M, Wall W, Huerta A (2020) A weakly compressible hybridizable discontinuous

Galerkin formulation for fluid-structure interaction problems. Comput Methods Appl Mech Eng 372:113,392
177. Lederer PL, Lehrenfeld C, Schöberl J (2018) Hybrid discontinuous Galerkin methods with relaxed H(div)-conformity for incompressible flows. Part I. SIAM J Numer Anal 56(4):2070–2094
178. Lederer PL, Lehrenfeld C, Schöberl J (2019) Hybrid discontinuous Galerkin methods with relaxed H(div)-conformity for incompressible flows. Part II. ESAIM Math Model Numer Anal 53(2):503–522
179. Lederer PL, Lehrenfeld C, Schöberl J (2020) Divergence-free tangential finite element methods for incompressible flows on surfaces. Int J Numer Methods Eng 121(11):2503–2533
180. Lee JJ, Shannon SJ, Bui-Thanh T, Shadid JN (2019) Analysis of an HDG method for linearized incompressible resistive MHD equations. SIAM J Numer Anal 57(4):1697–1722
181. Lehrenfeld C, Schöberl J (2016) High order exactly divergence-free hybrid discontinuous Galerkin methods for unsteady incompressible flows. Comput Methods Appl Mech Eng 307:339–361
182. Leng H, Chen Y (2020) Adaptive hybridizable discontinuous Galerkin methods for nonstationary convection–diffusion problems. Adv Comput Math 46(4):50
183. Li G, Shi K (2018) Upscaled HDG methods for Brinkman equations with high-contrast heterogeneous coefficient. J Sci Comput 77(3):1780–1800
184. Li L, Lanteri S, Mortensen NA, Wubs M (2017) A hybridizable discontinuous Galerkin method for solving nonlocal optical response models. Comput Phys Commun 219:99–107
185. Li L, Lanteri S, Perrussel R (2014) A hybridizable discontinuous Galerkin method combined to a Schwarz algorithm for the solution of 3D time-harmonic Maxwell's equation. J Comput Phys 256:563–581
186. Li L, Lanteri S, Perrussel R (2015) A class of locally well-posed hybridizable discontinuous Galerkin methods for the solution of time-harmonic Maxwell's equations. Comput Phys Commun 192:23–31
187. Liu Y (2009) Fast multipole boundary element method: theory and applications in engineering. Cambridge University Press, Cambridge
188. Loseille A, Feuillet R (2018) Vizir: high-order mesh and solution visualization using OpenGL 4.0 graphic pipeline. In: 2018 AIAA aerospace sciences meeting, p 1174
189. Lu P, Chen H, Qiu W (2017) An absolutely stable hp-HDG method for the time-harmonic Maxwell equations with high wave number. Math Comput 86(306):1553–1577
190. McLachlan RI, Stern A (2020) Multisymplecticity of hybridizable discontinuous Galerkin methods. Found Comput Math 20(1):35–69
191. Montlaur A, Fernández-Méndez S, Huerta A (2008) Discontinuous Galerkin methods for the Stokes equations using divergence-free approximations. Int J Numer Methods Fluids 57(9):1071–1092
192. Moon M, Lazarov R, Jun HK (2019) Multiscale HDG model reduction method for flows in heterogeneous porous media. Appl Numer Math 140:115–133
193. Moro D, Nguyen NC, Peraire J (2011) Navier–Stokes solution using hybridizable discontinuous Galerkin methods. In: 20th AIAA computational fluid dynamics conference. AIAA
194. Muixí A, Rodríguez-Ferran A, Fernández-Méndez S (2020) A hybridizable discontinuous Galerkin phase-field model for brittle fracture with adaptive refinement. Int J Numer Methods Eng 121(6):1147–1169
195. Muralikrishnan S, Bui-Thanh T, Shadid JN (2020) A multilevel approach for trace system in HDG discretizations. J Comput Phys 407:109,240
196. Muralikrishnan S, Tran M, Bui-Thanh T (2018) An improved iterative HDG approach for partial differential equations. J Comput Phys 367:295–321
197. Mustapha K, Nour M, Cockburn B (2016) Convergence and superconvergence analyses of HDG methods for time fractional diffusion problems. Adv Comput Math 42(2):377–393
198. Nelson B, Liu E, Kirby RM, Haimes R (2012) Elvis: a system for the accurate and interactive visualization of high-order finite element solutions. IEEE Trans Vis Comput Gr 18(12):2325–2334
199. Nguyen N, Peraire J, Cockburn B (2010) A hybridizable discontinuous Galerkin method for Stokes flow. Comput Methods Appl Mech Eng 199(9–12):582–597
200. Nguyen NC, Peraire J, Cockburn B (2009) An implicit high-order hybridizable discontinuous Galerkin method for linear convection–diffusion equations. J Comput Phys 228(9):3232–3254
201. Nguyen NC, Peraire J, Cockburn B (2009) An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations. J Comput Phys 228(23):8841–8855
202. Nguyen NC, Peraire J, Cockburn B (2011) High-order implicit hybridizable discontinuous Galerkin methods for acoustics and elastodynamics. J Comput Phys 230(10):3695–3718
203. Nguyen NC, Peraire J, Cockburn B (2011) Hybridizable discontinuous Galerkin methods for the time-harmonic Maxwell's equations. J Comput Phys 230(19):7151–7175
204. Nguyen NC, Peraire J, Cockburn B (2011) An implicit high-order hybridizable discontinuous Galerkin method for the incompressible Navier–Stokes equations. J Comput Phys 230(4):1147–1170
205. Nguyen NC, Peraire J, Cockburn B (2015) A class of embedded discontinuous Galerkin methods for computational fluid dynamics. J Comput Phys 302:674–692
206. Oikawa I (2015) A hybridized discontinuous Galerkin method with reduced stabilization. J Sci Comput 65(1):327–340
207. Oikawa I (2016) Analysis of a reduced-order HDG method for the Stokes equations. J Sci Comput 67(2):475–492
208. Paipuri M, Tiago C, Fernández-Méndez S (2019) Coupling of continuous and hybridizable discontinuous Galerkin methods: application to conjugate heat transfer problem. J Sci Comput 78(1):321–350
209. Peraire J, Nguyen NC, Cockburn B (2010) A hybridizable discontinuous Galerkin method for the compressible Euler and Navier–Stokes equations. AIAA Pap 363:2010
210. Peters E, Evans J (2019) A divergence-conforming hybridized discontinuous Galerkin method for the incompressible Reynolds-averaged Navier–Stokes equations. Int J Numer Methods Fluids 91:112–133
211. Pignet N (2019) Hybrid high-order methods for nonlinear solid mechanics. PhD thesis, Université Paris-Est Marne la Vallée. TEL 02318157
212. Poya R, Sevilla R, Gil AJ (2016) A unified approach for a posteriori high-order curved mesh generation using solid mechanics. Comput Mech 58(3):457–490
213. Prud'homme C (2006) A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations. Sci Program 14:150,736
214. Qiu W, Shen J, Shi K (2018) An HDG method for linear elasticity with strong symmetric stresses. Math Comput 87(309):69–93
215. Qiu W, Shi K (2016) A superconvergent HDG method for the incompressible Navier–Stokes equations on general polyhedral meshes. IMA J Numer Anal 36(4):1943–1967
216. Qiu W, Shi K (2019) Analysis on an HDG method for the p -Laplacian equations. J Sci Comput 80(2):1019–1032
217. Qiu W, Solano M, Vega P (2016) A high order HDG method for curved-interface problems via approximations from straight triangulations. J Sci Comput 69(3):1384–1407

218. Quarteroni A (2017) Numerical models for differential problems. MS&A modeling, simulation and applications, vol 16. Springer, Cham
219. Rathgeber F, Ham DA, Mitchell L, Lange M, Luporini F, Mcrae ATT, Bercea GT, Markall GR, Kelly PHJ (2016) Firedrake: automating the finite element method by composing abstractions. ACM Trans Math Softw 43(3):1–27
220. Remacle JF, Chevaugeon N, Marchandise E, Geuzaine C (2007) Efficient visualization of high-order finite elements. Int J Numer Methods Eng 69(4):750–771
221. Renard Y, Poulios K (2020) GetFEM: automated FE modeling of multiphysics problems based on a generic weak form language. Technical report, HAL. https://hal.archives-ouvertes.fr/hal-02532422
222. Rhebergen S, Cockburn B (2012) A space-time hybridizable discontinuous Galerkin method for incompressible flows on deforming domains. J Comput Phys 231(11):4185–4204
223. Rhebergen S, Wells G (2018) A hybridizable discontinuous Galerkin method for the Navier–Stokes equations with pointwise divergence-free velocity field. J Sci Comput 76(3):1484–1501
224. Rhebergen S, Wells G (2018) Preconditioning of a hybridized discontinuous Galerkin finite element method for the Stokes equations. J Sci Comput 77(3):1936–1952
225. Rhebergen S, Wells GN (2020) An embedded-hybridized discontinuous Galerkin finite element method for the Stokes equations. Comput Methods Appl Mech Eng 358:112,619
226. Rivière B (2008) Discontinuous Galerkin methods for solving elliptic and parabolic equations. Society for Industrial and Applied Mathematics, Philadelphia
227. Rocha BM, dos Santos RW, Igreja I, Loula AFD (2020) Stabilized hybrid discontinuous Galerkin finite element method for the cardiac monodomain equation. Int J Numer Methods Biomed Eng 36(7):e3341
228. Samii A, Dawson C (2018) An explicit hybridized discontinuous Galerkin method for Serre–Green–Naghdi wave model. Comput Methods Appl Mech Eng 330:447–470
229. Samii A, Kazhyken K, Michoski C, Dawson C (2019) A comparison of the explicit and implicit hybridizable discontinuous Galerkin methods for nonlinear shallow water equations. J Sci Comput 80(3):1936–1956
230. Samii A, Michoski C, Dawson C (2016) A parallel and adaptive hybridized discontinuous Galerkin method for anisotropic nonhomogeneous diffusion. Comput Methods Appl Mech Eng 304:118–139
231. Samii A, Panda N, Michoski C, Dawson C (2016) A hybridized discontinuous Galerkin method for the nonlinear Korteweg–de Vries equation. J Sci Comput 68(1):191–212
232. Sánchez MA, Ciuca C, Nguyen NC, Peraire J, Cockburn B (2017) Symplectic Hamiltonian HDG methods for wave propagation phenomena. J Comput Phys 350:951–973
233. Sánchez-Vizuet T, Solano ME (2019) A hybridizable discontinuous Galerkin solver for the Grad–Shafranov equation. Comput Phys Commun 235:120–132
234. Sánchez-Vizuet T, Solano ME, Cerfon AJ (2020) Adaptive hybridizable discontinuous Galerkin discretization of the Grad–Shafranov equation by extension from polygonal subdomains. Comput Phys Commun 255:107,239
235. Schöberl J (2014) C++11 implementation of finite elements in NGSolve. Technical Report, ASC-30/2014, Institute for Analysis and Scientific Computing, TU Wien. https://www.asc.tuwien.ac.at/~schoeberl/wiki/publications/ngs-cpp11.pdf
236. Schoeder S, Kronbichler M, Wall WA (2018) Arbitrary high-order explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. J Sci Comput 76(2):969–1006
237. Schoeder S, Sticko S, Kreiss G, Kronbichler M (2020) High-order cut discontinuous Galerkin methods with local time stepping for acoustics. Int J Numer Methods Eng 121(13):2979–3003
238. Schütz J, Aizinger V (2017) A hierarchical scale separation approach for the hybridized discontinuous Galerkin method. J Comput Appl Math 317:500–509
239. Sevilla R (2019) HDG-NEFEM for two dimensional linear elasticity. Comput Struct 220:69–80
240. Sevilla R, Borchini L, Giacomini M, Huerta A (2020) Hybridisable discontinuous Galerkin solution of geometrically parametrised Stokes flows. Comput Methods Appl Mech Eng 372:113,397
241. Sevilla R, Fernández-Méndez S, Huerta A (2008) NURBS-enhanced finite element method (NEFEM). Int J Numer Methods Eng 76(1):56–83
242. Sevilla R, Fernández-Méndez S, Huerta A (2011) 3D NURBS-enhanced finite element method (NEFEM). Int J Numer Methods Eng 88(2):103–125
243. Sevilla R, Giacomini M, Huerta A (2018) A face-centred finite volume method for second-order elliptic problems. Int J Numer Methods Eng 115(8):986–1014
244. Sevilla R, Giacomini M, Huerta A (2019) A locking-free face-centred finite volume (FCFV) method for linear elastostatics. Comput Struct 212:43–57
245. Sevilla R, Giacomini M, Karkoulias A, Huerta A (2018) A superconvergent hybridisable discontinuous Galerkin method for linear elasticity. Int J Numer Methods Eng 116(2):91–116
246. Sevilla R, Huerta A (2016) Tutorial on hybridizable discontinuous Galerkin (HDG) for second-order elliptic problems. In: Schröder J, Wriggers P (eds) advanced finite element technologies. CISM International Centre for Mechanical Sciences, vol 566. Springer, Berlin, pp 105–129
247. Sevilla R, Huerta A (2018) HDG-NEFEM with degree adaptivity for Stokes flows. J Sci Comput 77(3):1953–1980
248. Sheldon JP, Miller ST, Pitt JS (2016) A hybridizable discontinuous Galerkin method for modeling fluid–structure interaction. J Comput Phys 326:91–114
249. Shen J, Singler JR, Zhang Y (2019) HDG-POD reduced order model of the heat equation. J Comput Appl Math 362:663–679
250. Solano M, Vargas F (2019) A high order HDG method for Stokes flow in curved domains. J Sci Comput 79(3):1505–1533
251. Soon SC, Cockburn B, Stolarski HK (2009) A hybridizable discontinuous Galerkin method for linear elasticity. Int J Numer Methods Eng 80(8):1058–1092
252. Stanglmeier M, Nguyen NC, Peraire J, Cockburn B (2016) An explicit hybridizable discontinuous Galerkin method for the acoustic wave equation. Comput Methods Appl Mech Eng 300:748–769
253. Stenberg R (1990) Some new families of finite elements for the Stokes equations. Numer Math 56(8):827–838
254. Su W, Wang P, Zhang Y, Wu L (2019) A high-order hybridizable discontinuous Galerkin method with fast convergence to steady-state solutions of the gas kinetic equation. J Comput Phys 376:973–991
255. Terrana S, Nguyen NC, Bonet J, Peraire J (2019) A hybridizable discontinuous Galerkin method for both thin and 3D nonlinear elastic structures. Comput Methods Appl Mech Eng 352:561–585
256. Terrana S, Vilotte J, Guillot L (2017) A spectral hybridizable discontinuous Galerkin method for elastic-acoustic wave propagation. Geophys J Int 213(1):574–602
257. Vidal-Codina F, Martín-Moreno L, Ciracì C, Yoo D, Nguyen NC, Oh SH, Peraire J (2020) Terahertz and infrared nonlocality and field saturation in extreme-scale nanoslits. Opt Express 28(6):8701–8715

258. Vidal-Codina F, Nguyen N, Oh SH, Peraire J (2018) A hybridizable discontinuous Galerkin method for computing nonlocal electromagnetic effects in three-dimensional metallic nanostructures. J Comput Phys 355:548–565
259. Vidal-Codina F, Nguyen N, Peraire J (2018) Computing parametrized solutions for plasmonic nanogap structures. J Comput Phys 366:89–106
260. Vidal-Codina F, Nguyen NC, Giles MB, Peraire J (2015) A model and variance reduction method for computing statistical outputs of stochastic elliptic partial differential equations. J Comput Phys 297:700–720
261. Vidal-Codina F, Nguyen NC, Giles MB, Peraire J (2016) An empirical interpolation and model-variance reduction method for computing statistical outputs of parametrized stochastic partial differential equations. SIAM-ASA J Uncertain Quantif 4(1):244–265
262. Vieira LM, Giacomini M, Sevilla R, Huerta A (2020) A second-order face-centred finite volume method for elliptic problems. Comput Methods Appl Mech Eng 358:112655
263. Vila-Pérez J, Giacomini M, Sevilla R, Huerta A (2020) Hybridisable discontinuous Galerkin formulation of compressible flows. Arch Comput Methods Eng https://doi.org/10.1007/s11831-020-09508-z
264. Wang CY (1991) Exact solutions of the steady-state Navier–Stokes equations. Annu Rev Fluid Mech 23(1):159–177
265. Wildey T, Muralikrishnan S, Bui-Thanh T (2019) Unified geometric multigrid algorithm for hybridized high-order finite element methods. SIAM J Sci Comput 41(5):S172–S195
266. Williams DM (2018) An entropy stable, hybridizable discontinuous Galerkin method for the compressible Navier–Stokes equations. Math Comput 87(309):95–121
267. Woopen M, Balan A, May G, Schütz J (2014) A comparison of hybridized and standard DG methods for target-based *hp*-adaptive simulation of compressible flow. Comput Fluids 98:3–16
268. Woopen M, May G, Schütz J (2014) Adjoint-based error estimation and mesh adaptation for hybridized discontinuous Galerkin methods. Int J Numer Methods Fluids 76(11):811–834
269. Xie ZQ, Sevilla R, Hassan O, Morgan K (2013) The generation of arbitrary order curved meshes for 3D finite element analysis. Comput Mech 51:361–374
270. Yang Y, Shi K, Fu S (2019) Multiscale hybridizable discontinuous Galerkin method for flow simulations in highly heterogeneous media. J Sci Comput 81(3):1712–1731
271. Yoo D, Vidal-Codina F, Ciracì C, Nguyen NC, Smith DR, Peraire J, Oh SH (2019) Modeling and observation of mid-infrared nonlocality in effective epsilon-near-zero ultranarrow coaxial apertures. Nat Commun 10(1):4476

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.