用于 Gridap.jl 非协调混合多胞方法的自然语言框架

JORDI MANYER[†]、JAI TUSHAR[♯], 和 SANTIAGO BADIA[†]

Abstract. 混合有限元方法，例如可杂交间断伽辽金法、混合高阶方法和弱伽辽金法，已成为在一般多面体网格上求解偏微分方程的强大技术。尽管这些方法的数学渊源各不相同，但它们具有共同的计算结构，涉及混合离散空间、局部投影算子和静态凝聚。本文提出了一个全面的框架，用于在 Gridap有限元库中实现此类方法。我们引入了若干新的抽象，包括利用基于图的结构表示多面体网格、在任意网格实体上定义分片多项式空间、针对逐单元线性系统进行基于片的局部组装、高层次局部算子构造以及自动静态凝聚。这些抽象使混合方法能够简洁实现，同时借助 Julia 的即时编译和 Gridap 的惰性求值策略保持计算效率。我们通过实现若干非协调多面体方法，展示了该框架在多面体网格上的泊松问题、线弹性、不可压缩斯托克斯流和最优控制中的应用。

arXiv:2603.00880v1 [cs.MS] 1 Mar 2026

1. Introduction

混合有限元（FE）方法，包括可杂交间断伽辽金（HDG）、混合高阶方法（HHO）、非协调虚拟元方法（VEM）和弱伽辽金（WG）方法，已成为在一般多面体网格上求解偏微分方程（PDE）的强大数值方法类别 [15, 16, 26, 30]。这些方法之所以特别具有吸引力，有以下几个原因。首先，它们支持在一般多边形和多面体网格上进行任意阶近似，因而能灵活采用网格生成以及局部加密和粗化策略。其次，它们在保持局部守恒性质的同时，实现最优收敛率。第三，它们天然支持静态凝聚，可高效消去基于单元的自由度（DoFs），从而得到仅定义在网格骨架上的约化全局系统。

尽管混合方法的起源和数学表述各不相同，但它们共享一种显著的计算结构：均依赖于将体积未知量与骨架（基于面）未知量配对的离散空间，采用通过逐单元线性求解计算的局部投影算子，由局部贡献组装全局系统，并利用静态凝聚缩减系统规模。这种结构上的相似性表明，统一的软件框架能够支持多种混合方法的实现，而无需编写针对特定方法的底层代码。

然而，在现有有限元库中实现混合多面体方法面临着重大挑战。大多数库都是围绕单纯形或六面体网格上的协调方法设计的，缺乏对一般多胞体的原生支持。因此，尽管近年来对混合方法（尤其是混合间断伽辽金）的支持有所增加，但通常仅限于协调的单纯形网格和六面体网格。广泛使用的库，例如 Deal.II [4], MFEM [2], FEniCS[24], NGSolve [27]和 DUNE [13]，就是如此。不过，近期已有一些工作致力于扩展其中部分库，使其支持多面体网格，例如 [22]。

因此，多面体能力更常见于 lymph[3], PolyDiM [11] 和 VEM++ [17] 等专用软件包。此外，混合方法的计算模式——尤其是局部算子组装和静态凝聚——需要标准有限元框架未提供的抽象。因此，实现这些方法的研究人员往往不得不编写自定义的底层代码，而这类代码难以维护、扩展和复用。

[†]School of mathematics, Monash university, Clayton, Victoria 3800, Australia
[♯]Center for Computation and Technology and Department of Mathematics, Louisiana State University Baton Rouge USA. , ,
E-mail addresses: jordi.manyer@monash.edu, jai.tushar@lsu.edu, santiago.badia@monash.edu.*Date*:2026年3月3日。

本文通过提出一个用于非协调混合多胞方法的自然语言框架，解决了这些挑战；用于离散给定偏微分方程的用户代码简洁且抽象层次高，与数学符号紧密对应。我们通过对 Gridap 有限元库进行一系列全面扩展来实现这一目标 [10, 29]。Gridap 是一个基于 Julia 的框架，利用即时编译（JIT）和多重分派，在避免传统双语言问题的同时实现高性能和可扩展性。在 Gridap 的基础上，我们引入了用于多面体网格表示、任意网格实体上的分片多项式空间、基于网格片的局部组装框架、局部算子构造以及自动静态凝聚的新抽象。

本研究的主要贡献如下：

- 利用循环图和旋转系统，为一般多边形和多面体构建基于图的表示，从而能够高效存储任意多胞体并进行拓扑查询。• 构建统一框架，用于在多面体网格的任意维切片上构造分片多项式空间，支持间断伽辽金（DG）、混合间断伽辽金、HHO 和相关方法所需的离散空间。•提供灵活的网格片组装基础设施，用于在任意网格实体网格片上集成并求解局部有限元问题，应用于局部投影算子和静态凝聚。• 为局部算子提供高层抽象，将通过逐单元线性求解计算离散投影的模式封装起来，从而隐藏实现复杂性。• 提供自动静态凝聚过程，在局部消去单元未知量以生成约化的骨架系统，并利用混合形式天然的分块结构。

这些抽象通过泊松问题的间断伽辽金、混合间断伽辽金和 HHO 方法完整实现，以及用于线弹性、不可压缩斯托克斯流和最优控制的 HHO 离散，得以展示。所得实现十分简洁——完整方法通常仅需 30–50 行代码——同时借助 Gridap 的惰性求值和 Julia 的即时编译保持了较高的计算效率。

本文其余部分安排如下。第 2 节以泊松问题为例，详细阐述混合高阶方法，并着重介绍混合方法共有的计算构件。第 3 节简要介绍 Gridap 库，并指出实现混合多面体方法所需的扩展。第 4 节介绍这些扩展的实现，包括多面体表示、不连续空间、局部组装框架和静态凝聚。第 5 节通过复杂度逐步提高的数值算例展示该框架。最后，我们总结本文的贡献，并展望未来的研究方向。

2. A motivating example: The Poisson problem with HHO

为明确问题并展示该库提供的抽象，我们考虑以下定义在多面体利普希茨区域 $\Omega \subset \mathbb{R}^D$ 及其边界 $\partial\Omega$ 上的泊松问题：求

$$u \in H^1(\Omega) \ : \ -\Delta u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega, \tag{2.1}$$

给定源项数据 $f \in L^2(\Omega)$ （为避免阐述中的一些技术细节）和边界数据$g \in H^{\frac{1}{2}}(\partial\Omega)$。其弱形式为：求

$$u \in H^1_g(\Omega) \ : \ a(u,v) = l(v) \quad \forall v \in H^1_0(\Omega), \tag{2.2}$$

其中 $H^1_0(\Omega)$ 以及 $H^1_g(\Omega)$ 表示标准索伯列夫空间，其中的函数属于 $H^1(\Omega)$ 使得其迹在 $\partial\Omega$ 上为零以及 g，分别为，并且双线性型与线性型定义为 $a(u, \ v) \doteq \int_\Omega \nabla u \cdot \nabla v, \quad l(v) \doteq \int_\Omega fv.$

我们现在使用 HHO 方法对该问题进行离散，并紧密遵循 [26, 第 2]章。这一阐述将突出该方法的关键特征，以及更广泛的混合方法族所共享的计算基本构件。

2.1. 混合离散空间。 给定由有限个互不重叠的多面体单元 T构成的 Ω 的多面体网格 $\mathcal{M}_h$ ，记网格单元集合为 $\mathcal{T}_h$ ，网格面集合为 $\mathcal{F}_h$ 。对于多项式次数 $k, l \geqslant 0$，HHO 方法引入如下混合离散空间：

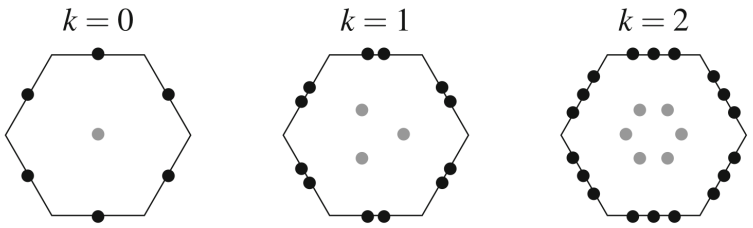

FIGURE 1. 六边形单元 T 上局部混合高阶方法离散空间的示意图，多项式次数为 $(k,l)=(0,0)$, $(1,1),(2,2)$。该空间由定义在单元上的多项式（灰色）以及定义在每个面上的多项式（黑色）组成。

$$\underline{U}_h^{k,l} \doteq \mathbb{P}^k(\mathcal{T}_h) \times \mathbb{P}^l(\mathcal{F}_h), \tag{2.3}$$

其中，$\mathbb{P}^k(\mathcal{T}_h)$ 和 $\mathbb{P}^l(\mathcal{F}_h)$ 分别表示网格单元和网格面上次数为k 和 l 的全局不连续分片多项式空间。因此，元素 $u_h \in U^{k,l}_h$ 是一个元组 $u_h = ((u_T)_{T\in\mathcal{T}_h}, (u_F)_{F\in\mathcal{F}_h})$，其中 u_T表示单元上的多项式，u_F 表示面（骨架）上的多项式。这种混合结构——将体自由度与网格骨架上的自由度配对——是混合间断伽辽金、混合高阶方法、弱伽辽金及相关方法共有的第一个特征。每个单元 T 上的局部空间如图 1 所示。定义混合离散空间后，下面介绍在这些空间之间进行映射的局部算子。

2.2. 局部算子与单元积分。 HHO 方法依赖于定义在每个单元 T 及其周围面上的局部算子。这些算子通过求解涉及单元及其面上积分的逐单元线性系统来构造——这是混合方法的第二个共同特征。

其主要构成模块是局部重构算子 $\mathcal{R}_T\colon U^{k,l}_h(T) \to \mathbb{P}^{k+1}(T)$，它在 $\underline{T}$ 内根据局部迹未知量 $\underline{v}_T = (v_T, (v_F)_{F\in\mathcal{F}_h(T)}) \in U^{k,l}_h(T)$计算高阶多项式近似。该算子由

$$\int_T \nabla\mathcal{R}_T\underline{v}_T \cdot \nabla w = \int_T \nabla v_T \cdot \nabla w + \sum_{F\in\partial T}\int_F (v_F - v_T)\nabla w \cdot \boldsymbol{n}_{TF} \quad \forall w \in \mathbb{P}^{k+1}(T), \tag{2.4}$$

$$\int_T \mathcal{R}_T\underline{v}_T = \int_T v_T, \tag{2.5}$$

其中，$\boldsymbol{n}_{TF}$表示从单元 T指向面 F的单位外法向量。计算 $\mathcal{R}_T v_T$需要组装一个小型局部线性系统，其中涉及在 T 及其各个面 $F \in \mathcal{F}_h(T)$上的积分，然后求解该系统以确定重构多项式的系数。

此外，该方法采用投影到单元和面上的 L^2-正交投影算子，即 $\pi^{0,k}_T\colon L^2(T) \to \mathbb{P}^k(T)$ 和 $\pi^{0,l}_F\colon L^2(F) \to \mathbb{P}^l(F)$，使得

$$\int_T \pi^{0,k}_T w\, v = \int_T w\, v \quad \forall v \in \mathbb{P}^k(T), \quad \text{and} \quad \int_F \pi^{0,l}_F w\, v = \int_F w\, v \quad \forall v \in \mathbb{P}^l(F). \tag{2.6}$$

这些投影算子也可以通过涉及单元或面积分的小型系统在局部计算。我们还定义所谓的差分算子 $\delta^k_T\colon U^{k,l}_h(T) \to \mathbb{P}^k(T)$和 $\delta^l_{TF}\colon U^{k,l}_h(T) \to \mathbb{P}^l(F)$，定义如下：

$$\delta^k_T\underline{v}_T = \pi^{0,k}_T(\mathcal{R}_T\underline{v}_T - v_T), \quad \text{and} \quad \delta^l_{TF}\underline{v}_T = \pi^{0,l}_F(\mathcal{R}_T\underline{v}_T - v_F).$$

请注意，局部算子（例如弱伽辽金方法中的弱梯度）也可以在此框架中轻松计算。有了这些局部算子，我们现在可以构造离散问题并描述组装过程。

2.3. 单元局部组装与静态凝聚。 全局离散问题由单元局部贡献组装而成，这是混合方法的第三个定义特征。记 $\mathcal{F}^{\mathrm{bd}}_h$为边界面的集合，方程 (2.2) 的高阶混合有限元近似如下：求 $u_h \in U^{k,l}_h$，$g \doteq \{u_h \in U^{k,l}_h\colon u_F =$

$\pi^{0,l}_F(g)\ \forall F \in \mathcal{F}_h^{\mathrm{bd}}\}$ 使得

$$a_h(\underline{u}_h, \underline{v}_h) = l_h(\underline{v}_h) \quad \forall \underline{v}_h \in \underline{U}_{h,0}^{k,l} \doteq \{\underline{v}_h \in \underline{U}_h^{k,l} : v_F = 0\ \forall F \in \mathcal{F}_h^{\mathrm{bd}}\}, \tag{2.7}$$
$$\text{where} \quad a_h(\underline{u}_h, \underline{v}_h) = \sum_{T\in\mathcal{T}_h} \left[a_T(\underline{u}_T, \underline{v}_T) + S_T(\underline{u}_T, \underline{v}_T)\right], \quad l_h(\underline{v}_h) = \sum_{T\in\mathcal{T}_h} l_T(\underline{v}_T),$$

其中 $\underline{u}_T = \underline{u}_h|_T$ 表示 u_h 在单元 T 及其周围面上的限制。局部双线性形式 $a_T(u_T, v_T)$ 表示一致性项，$S_T(u_T, v_T)$ 提供稳定化，$l_T(v_T)$ 为局部载荷项。所涉及的局部形式定义为

$$\begin{aligned} a_T(\underline{u}_T, \underline{v}_T) &= \int_T \nabla\mathcal{R}_T\underline{u}_T \cdot \nabla\mathcal{R}_T\underline{v}_T, \\ S_T(\underline{u}_T, \underline{v}_T) = h_T^{-1} \sum_{F\in\partial T} \int_F (\delta^l_{TF} - \delta^k_T)\underline{u}_T \cdot (\delta^l_{TF} - \delta^k_T)\underline{v}_T, &\quad \text{and} \quad l_T(\underline{v}_T) = \int_T f v_T. \end{aligned} \tag{2.8}$$

其中 h_T表示单元 T的直径。在定义 S_T的面积分中，单元多项式$\delta^k_T v_T \in \mathbb{P}^k(T)$ 被理解为限制在 F上。对这些形式求值需要在单元 T 及其面上积分，从而为每个单元生成一个局部矩阵和一个向量，将单元未知量 u_T与面未知量 $(u_F)_{F\in\mathcal{F}_h(T)}$耦合起来。

混合方法的计算效率源于静态凝聚：由于单元未知量仅出现在各自的局部贡献中，它们可以在局部被消去，从而得到仅涉及面未知量的约化全局系统。该凝聚过程包括：

(1) 对每个单元 T，组装耦合单元自由度与面自由度的局部系统；(2) 根据面未知量在局部求解单元未知量（舒尔补运算）；(3) 仅在网格骨架 $\mathcal{F}_h$ 上组装约化全局系统。

约化系统通常比完整系统小得多，尤其是在多项式次数较高时；求解完成后，可以通过局部求解高效地恢复单元未知量。

2.4. 迈向统一的计算框架。 前文对泊松问题中混合高阶方法的阐述，揭示了一种计算结构，这种结构是更广泛的混合方法（混合间断伽辽金、混合高阶方法、弱伽辽金）所共有的。从实现角度来看，这些方法需要相同的基本构件：

(1) **混合离散空间：** 自由度分布在不同维度的网格实体上——通常是单元和面——形成积空间结构。实现时必须处理数据结构，以便将未知量与体积单元和网格骨架关联起来。(2) **单元和面上的局部积分：** 所有方法都需要计算每个单元 T 及其边界面 $F \in \mathcal{F}_h(T)$上的积分。这些积分出现在局部算子（例如重构、投影）的定义中，也用于计算局部双线性型和线性型。实现时必须提供适用于一般多胞体的高效求积规则和基函数求值方法。(3) **局部算子的组装与求解：** 许多混合方法定义了辅助算子（重构、投影、稳定化），这些算子通过在每个单元上组装并求解小型局部线性系统来计算。框架必须提供机制，能够根据用户定义的变分形式组装这些系统并高效求解。(4) **单元局部组装与静态凝聚：** 全局离散系统通过遍历各个单元进行组装，计算耦合单元与面未知量的局部贡献，然后凝聚掉单元未知量，在网格骨架上形成约化全局系统。实现必须自动完成这一凝聚过程，并管理约化系统的组装。

尽管混合方法种类繁多，其计算工作流却惊人地统一：对每个单元积分局部形式，求解局部系统以计算辅助算子，组装局部矩阵和向量，执行静态凝聚，并将结果贡献到全局骨架系统中。这种统一性表明，可以设计一个通用计算框架，以支持实现广泛的混合方法，而无需为每种变体编写底层的方法特定代码。

本工作的目标是介绍这样一个框架。该框架构建于 Gridap 有限元库之上，能够抽象出通用的计算模式，同时保持足够的灵活性，以适应

不同混合方法的具体特点。通过为混合空间、局部算子和静态凝聚提供高层抽象，该框架使用户能够以最少的工作量实现复杂的混合离散化，从而专注于数学表述，而非底层实现细节。值得一提的是，标准间断伽辽金方法也可作为一种特殊情况纳入其中：省略面未知量、局部重构算子和静态凝聚后，该框架便会简化为传统的间断伽辽金组装。接下来的章节将介绍该框架的设计与实现，并展示其在若干混合方法中的应用。

3. Introduction to Gridap.jl

Gridap [10, 29] 是一个完全使用 Julia 编程语言编写的开源有限元框架[12]，用于偏微分方程的数值近似。Gridap 完全使用 Julia 编写，通过利用 Julia 的即时编译，在运行时生成高效且针对具体问题的机器码，从而避免了科学计算中常见的双语言问题——即用 C++ 或 Fortran 编写对性能要求高的代码，同时用 Python 提供用户界面。这样，研究人员便能完全使用高级语言扩展该库并开发新的数值方法，而不牺牲性能。

与采用传统面向对象设计（如 Deal.II [4]）或依赖领域专用形式编译器（如 FEniCS [24]）的有限元库不同，Gridap 采用基于 Julia 多重分派范式和惰性求值的软件架构。用户无需显式遍历网格单元，而是构建表示所有单元上单元量（如刚度矩阵或载荷向量）的对象。这些对象通常是惰性的：底层数据会在需要时即时计算，而不是预先将所有单元的数据存储在内存中，从而降低内存需求。这种设计将组装循环隐藏在用户代码中，使语法简洁且与数学弱形式十分相似——只需 10–20 行代码即可求解标准的泊松问题或斯托克斯问题。

Gridap 还提供一种模块化的底层接口，它基于少量源自数学概念的基本抽象：数组（AbstractArray）、可调用对象（Map）、物理场（Field）、张量值（MultiValue）和线性泛函（Dof）。这些抽象可以通过多重分派进行组合和扩展，使高级用户无需修改库的核心即可自定义其行为。 [29], 中对这些软件抽象作了全面说明， [23]中则提供了相关教程。

4. New features for hybrid polytopal methods

本节介绍为支持混合多面体方法而添加到 Gridap 中的新抽象。我们首先说明如何使用图结构表示一般多胞体（第 4.1 节），接着介绍多面体网格及其拓扑查询（第 4.2 节）。随后，我们介绍任意网格切片上的分片多项式空间（第 4.3 节），再说明局部组装与求解框架（第 4.4 节）、局部算子抽象（第 4.5 节）以及静态凝聚过程（第 4.6 节）。此处介绍的所有功能均可在 Gridap v0.19.8 及更高版本中使用。

4.1. 多胞体的一般表示。我们遵循 [9]中采用的框架，使用旋转系统表示一般多胞体——旋转系统是平面图的一个子集，其中每个顶点的邻点都按循环序排列。这些基于图的表示能够实现高效存储、灵活的拓扑查询，并可轻松集成到我们的有限元装配流水线中。为求完整起见，我们在此重述主要结果。

对于给定的图 G， G的顶点集记为 $\mathsf{vert}(G)$，其邻接关系用 $\mathsf{adj}(G)$表示。与顶点 $\alpha \in \mathsf{vert}(G)$ 相邻的顶点集记为 $\mathsf{adj}(G)(\alpha)$。在旋转系统 R中，对于顶点 $\alpha \in \mathsf{vert}(R)$ 和一个相邻顶点 $\beta \in \mathsf{adj}(R)(\alpha)$，由于循环排序，$\mathsf{adj}(R)(\alpha)$中存在明确定义的前驱顶点和后继顶点。因此，我们可以将 $\mathsf{next}(\alpha, \beta)$ 定义为在循环序 $\mathsf{adj}(R)(\alpha)$中紧随 β 的顶点。

多胞体 $P \subset \mathbb{R}^d$ 定义为一种旋转系统，其顶点是 $\mathbb{R}^d$中的点。在二维中，每个边界顶点恰有两个邻点，因此旋转系统简化为一个简单闭回路，从而还原出经典的多边形表示。在三维中，循环序编码了每个顶点周围的局部面结构，而 P 的边界则构成一个由多边形组成的定向闭曲面。图 2 展示了盾状体的这三种表示。盾状体是一种近期在活细胞中发现的多面体，有助于活细胞高效地填充弯曲组织。

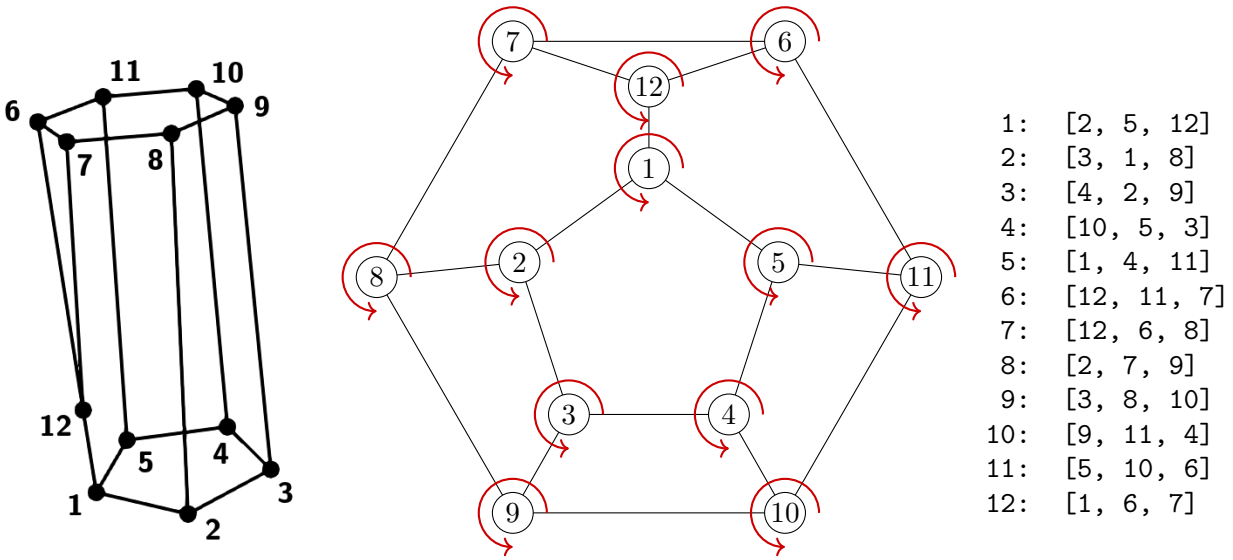

FIGURE 2. 盾状体的不同表示。左：三维多面体表示。中：关联的旋转系统 R。右：顶点邻接表 adj(R)。

一般多胞体

```
using
using       .
using       .           :
using       .           :

# 2D polygons - Regular n-gons
## Uniform points on a circle
function     (
    ::Point{D}, ::Integer, ::Real=0.
) where D
    ( ) = 2 *( -1)/
    ,   =    ( ),    ( )
      =      (      ( ==2,( , ),( , ,0.)))
      =      (      ( ==2,(- , ),(- , ,0.)))
   [  +    ( ( ))*   +    ( ( ))*   for   in 1: ]
end
          =        (    (     (0.,0.),6))
          =        (    (     (0.,0.),7))

# 3D polyhedra - Scutoid
## Coordinates
  = - /6 # Twist angle
  =     (
      (     (0.,0.,0.),5, /2)...,    # Bottom
      (     (0.,0.,4.),6, /2+ )...,  # Top
       (0.0,1.0,1)                   # Apex
)


## Polyhedron via vertex adjacency list
      = Vector{Int32}[
 [2, 5, 12], [3, 1, 8],   [4, 2, 9],  [10, 5, 3],
 [1, 4, 11], [12, 11, 7], [12, 6, 8], [2, 7, 9],
 [3, 8, 10], [9, 11, 4],  [5, 10, 6], [1, 6, 7],
]
  =           ( ,    )

## Polyhedron via faces
  = Vector{Int32}[
  [5,4,3,2,1], [6,7,8,9,10,11], [4,5,11,10],
  [5,1,12,6,11], [12,7,6], [12,1,2,8,7],
  [8,2,3,9], [9,3,4,10]
]
,   =                      ( , )

# API
  =                      ( ) # Vertex coordinates

## get_faces(P,d_from,d_to)
    =          ( ,2,0) # Face to node connectivity
    =          ( ,1,0) # Edge to node connectivity
    =          ( ,2,1) # Face to edge connectivity

  =          ( ) # Vertex adjacency list
```

LISTING 1. Gridap 中一般多胞体的构造函数。

代码清单 1 展示了如何在 Gridap 中构造一般多胞体。在二维中，多边形由按序排列的顶点坐标列表构造（第 17–18 行）；这些顶点的循环序隐式编码了旋转系统，因为在二维中，旋转系统简化为一个简单闭回路。在三维中，多面体可以通过两种等价方式指定，示例均采用图 2 所示的盾状体：一种是直接使用顶点邻接表 adj(R)（第 29–35 行），这与上文介绍的旋转系统 R 完全对应；另一种是通过 polyhedron from faces 枚举定向面（第 37–43 行），Gridap 会据此自动推断旋转系统。构造完成后，可通过 get faces(P, d, s) 提取拓扑信息（第 49–51 行）；对于 P的每个 d-面，该函数会返回其关联的 s-面列表，例如每个二维面的顶点索引，或界定每个面的边。底层旋转系统的邻接表 adj(R) 也可以通过 get graph(P) 直接获取（第 53 行）。

沃罗诺伊网格

```
using
using        .

function              (     ...)
              =                        (     ...) |>
         =          .        (          )
  return
end

       =              ((0,1,0,1),(6,6))
         (      ,          )
```

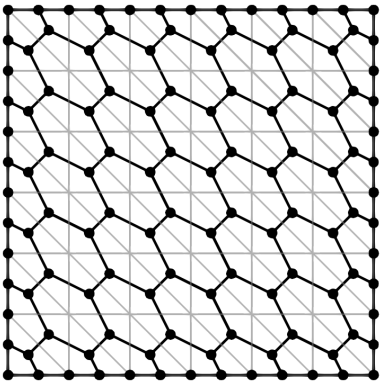

FIGURE 3. 使用 Gridap 生成二维沃罗诺伊网格。左：用于生成沃罗诺伊网格的 Julia 代码。右：生成的网格可视化。初始单纯形化笛卡尔网格以浅灰色显示，供参考。

4.2. 多面体网格。传统的单纯形网格或六面体网格可以通过存储一个参考多面体、顶点坐标列表以及一张将每个单元映射到其顶点的连通表来高效表示。利用这些信息，可以计算几何映射，将参考多面体映射到每个实际多面体上。这种方法通过避免冗余存储几何信息，最大限度地减少了内存使用。

对于一般的多面体网格，这种策略因两个根本原因而行不通。首先，每个多面体都可能是独特的，不存在一个统一的参考单元。其次，一般而言，无法实际定义从一个一般多面体到另一个拓扑相同的一般多面体的几何映射。因此，一般的多面体网格需要新的数据结构；这些结构虽然设计起来更直接，但可能比传统结构更占用内存。

为此，我们提供了一种新的 `PolytopalGridTopology` 结构，用于存储表示多面体网格拓扑所需的全部信息。随后，该拓扑会与一个 `FaceLabeling`配对，由它为网格的每个面分配物理标签，从而生成一个 `PolytopalDiscreteModel` 对象。目前，我们为`PolytopalDiscreteModel`提供了一个底层构造函数，以及若干用于生成多面体网格的高层方法：从二维背景单纯形网格进行沃罗诺伊剖分（如图 3 所示），以及通过构造函数`PolytopalDiscreteModel(model)`直接转换二维和三维中已有的单纯形网格或四边形网格。尽管本文未对此进行演示，该框架还支持构造任意聚合的网格 [7]。

现在，我们描述如何从多面体网格中提取拓扑信息。考虑区域 $\Omega \subset \mathbb{R}^D$ 的多面体网格 $\mathcal{M}_h \subset \mathbb{R}^D$。对于任意维数 $0 \leqslant d \leqslant D$，我们将网格中的 d 维实体称为 d 面。特别地，D 面对应于网格单元（或元素），而 $(D-1)$ 面对应于网格边面。

我们用 $\mathcal{F}_h^d$表示 $\mathcal{M}_h$的 d-面集合。特别地，有 $\mathcal{T}_h \doteq \mathcal{F}_h^D$和 $\mathcal{F}_h \doteq \mathcal{F}_h^{D-1}$。我们将用 T、F和 F^d分别表示 $\mathcal{T}_h$、$\mathcal{F}_h$和 $\mathcal{F}_h^d$中的一般元素。

G给定一个 d-面 $F^d \in \mathcal{F}_h^d$，我们用 $\mathcal{F}_h^s(F^d)$表示围绕 F^d的 s-面集合，其定义为 s

$$\mathcal{F}_h^s(F^d) \doteq \begin{cases} \{F^s \in \mathcal{F}_h^s : F^s \subset \overline{F^d}\} & \text{if } 0 < s < d, \\ \{F^d\} & \text{if } s = d \\ \{F^s \in \mathcal{F}_h^s : F^d \subset \overline{F^s}\} & \text{if } d < s \leqslant D. \end{cases} \tag{4.1}$$

特别地，F^d的边界由 $\partial F^d \doteq \mathcal{F}_h^{d-1}(F^d)$给出。例如，单元 T的边界由其面组成：$\partial T = \mathcal{F}_h^{D-1}(T)$。在 Gridap 中，可以通过 get faces(object, d, s) API 获取此信息；该 API 在多胞体层级（见列表 1）和网格层级均有定义。有关更详细的说明，请参阅提供的教程 [23] 。

4.3. 分片多项式空间。在说明如何表示和查询多面体网格拓扑后，我们现在介绍定义在这些网格上的有限元空间。所有间断方法和混合非协调方法都依赖于定义在网格 d 维面上的分片多项式空间。对于本文介绍的混合方法，我们只处理多项式

单元上的空间（$d = D$）和面上的空间（$d = D - 1$）。不过，该实现支持在任意维数的面上使用这套工具，从而为更复杂的构造铺平道路，例如协调虚单元方法 [16] 或多面体网格上的离散 de Rham 序列 [18]。

我们首先介绍局部多项式空间。记 $\mathbb{P}^k$ 为在 $\mathbb{R}^d$上次数不超过 k 的多项式空间。设 $X \subset \mathbb{R}^d$，$d \geqslant 1$为开有界连通集。局部多项式空间 $\mathbb{P}^k(X)$定义为 $\mathbb{P}^k$在 X上的限制。给定 $\mathcal{X}_h \subset \mathcal{F}_h^d$，我们可以在 $\mathcal{X}_h$ 上定义分片多项式空间，即由在每个 d-面 $F^d \in \mathcal{X}_h$上均为多项式的函数构成的集合，具体如下：

$$\mathbb{P}^k(\mathcal{X}_h) = \bigtimes_{F^d \in \mathcal{X}_h} \mathbb{P}^k(F^d). \tag{4.2}$$

我们提供了一种新的多面体有限元空间类型，用于在多面体网格的各个切片上构造任意类型的分片多项式空间，如列表 2 所示。第 10–11 行首先根据离散模型创建单元和面三角剖分 $\Omega \doteq \mathcal{T}_h$ and $\Gamma \doteq \mathcal{F}_h$。随后，通过提供三角剖分、值类型和多项式阶数 k，将向量值分片单元空间 $[\mathbb{P}^k(\mathcal{T}_h)]^D$ 构造为 V（第 14–16 行）。这种构造可以推广到 $\mathbb{P}^k(F^d)$ 的任意局部多项式子空间。具体而言，对于

多面体空间

```
1  using
2  using       .
3  using       .
4  using       .
5
6        =             ((0,1,0,1),(6,6))
7
8        = 1
9    =               (     )
10   =               (ReferenceFE{D},      )
11   =               (ReferenceFE{D-1},      )
12
13 # [Pᵏ(T)]ᵖ
14   =                  (
15    , VectorValue{D,Float64},
16 )
17
18 # Pᵏ(T) \ Pᵖ(T)
19   =                  (
20    , Float64,      ;
21                =
22 )
23
24 # Pᵏ(F), orthonormalized and zero on ∂Ω
25   =                  (
26    , Float64,      ;
27                  =           ,
28                =     ,
29               =
30 )
31
32 # Add Dirichlet BCs to M on ∂Ω
33  ( ) =  [1]^2 +  [2]^2
34   =              ( ,  )
35
36 # [Pᵏ(F)]^{DxD,sym}
37   =                  (
38    , SymTensorValue{2,Float64},
39 )
```

Listing 2. 使用多面体有限元空间构造函数，在多面体网格上构造分片多项式有限元空间。此示例展示了如何在不同网格实体（单元、面）上创建具有不同值类型和多项式次数的空间。

$\mathbb{X}(F^d) \subset \mathbb{P}^k(F^d)$，我们将其对应的分片多项式空间定义为

$$\mathbb{X}(\mathcal{X}_h) = \bigtimes_{F^d \in \mathcal{X}_h} \mathbb{X}(F^d). \tag{4.3}$$

文献中经常会出现这类空间的示例，例如在每个 d-面 $F^d \in \mathcal{X}_h$ 上定义的、次数至多为 k 的局部零均值多项式空间，其定义为

$$\mathbb{P}_0^k(F^d) = \left\{ p \in \mathbb{P}^k(F^d) : \int_{F^d} p \, d\boldsymbol{x} = 0 \right\}. \tag{4.4}$$

这个零均值子空间通过将局部核 =:constants 传递给多面体有限元空间来构造，如 W（第 19–22 行）所示；该空间会从每个局部基中投影掉局部常数分量。我们将在第 5.2.2 节中使用这一特性，为斯托克斯问题定义局部零均值压力空间。

分片空间同样可以定义在面网格 $\Gamma \doteq \mathcal{F}_h$ 上。第 25–30 行将标量面空间 $\mathbb{P}^k(\mathcal{F}_h)$ 构造为 M，并通过狄利克雷标记在 $\partial\Omega$ 上施加狄利克雷边界条件，还可选择使其同时具有局部层次性和正交性。-这两个性质对于混合高阶方法中的骨架空间通常很有用，有助于在存在小面 [6] 时保持收敛率。随后，通过标准的 TrialFESpace 构造函数（第 33–34 行），即可得到具有指定狄利克雷数据 u 的相应试探空间。最后，L（第 37–39 行）通过构造对称张量值面空间 $[\mathbb{P}^k(\mathcal{F}_h)]^{D\times D,\,\mathrm{sym}}$，展示了多面体有限元空间能够处理任意值类型。

这些间断多项式空间既可作为混合方法的试探空间和测试空间，也可作为重构目标；接下来我们将介绍局部算子。

4.4. 局部组装与求解。 组装并求解局部（通常按单元划分）的线性系统是混合方法的核心。例如，这些操作可用于静态凝聚单元自由度，或计算局部投影，例如 HHO 重构算子。作为扩展该库的一部分，我们提供了一个灵活的高级框架，用于对定义在网格几何实体子集上的局部有限元问题进行积分、组装和求解，这些子集称为网格片。

尽管本文所需的所有局部求解均定义在单个单元及其周围的面上，我们仍将代码设计为支持任意网格片定义。举例来说，该框架可处理多单元网格片、位于共享多个面的单元之间的界面网格片，以及存在重叠的网格片（即一个网格实体属于多个网格片）。这种灵活性使其能够应用于混合方法以外的领域，例如几何多重网格求解器中的基于网格片的平滑器，这与 [21] 中开发的 PCPATCH 框架类似。GridapSolvers 软件包 [25]中提供了该框架用于多重网格求解器的示例。不过，这类应用超出了本文的范围。

我们的框架围绕三个新对象构建。首先，PatchTopology 为每个待求解的局部问题定义需要考虑其贡献的一组网格实体（补丁）。其次，PatchTriangulation 对象提取这些补丁的 D 维切片，以便在其中定义并积分不同的贡献。最后，PatchAssembler 收集来自不同 PatchTriangulation 对象的所有贡献，并将它们组装成局部矩阵、向量或矩阵—向量对（局部线性系统）的数组。

PatchTopology 的定义方式是：提供一个底层的 GridTopology 对象，并针对每个维度，列出属于各个补丁的 d 面。例如，在清单 3（第 29 行）中，我们为多面体网格 $\mathcal{M}_h$ 的每个单元 T 定义周围的补丁，其中每个补丁都包含单元 T 本身及其周围的所有面，即 $\{T,\ (\mathcal{F}_h^d(T))_{d<D}\}$。

Gridap 中的积分围绕 Triangulation 对象展开，这类对象表示网格的单维切片（即固定 d 下的一组 d 面），并在其中定义积分贡献。我们为补丁上的 Triangulation 对象提供了两种构造函数：PatchTriangulation 包含每个补丁的 D 面，而 PatchBoundaryTriangulation 则包含每个补丁中位于补丁边界上的 $(D-1)$ 面。内部会将这些面拼接起来以提高积分效率，同时保留足够的信息以支持逐补丁积分。清单 3（第 30–31 行）展示了如何为上述补丁创建这两种类型的 Triangulation 对象，并将其用于定义 HHO 重构算子 (2.4) 的约束局部右端项（RHS）和左端项（LHS）（第 35–40 行）。

最后，我们提供一个 PatchAssembler 对象，负责将定义在不同 PatchTriangulation 对象上的局部贡献组装为局部数组。它同时支持单变量问题和多变量问题，并且在后一种情况下允许局部块组装。清单 3（第 43-47 行）展示了如何使用 PatchAssembler 来组装 (2.4) 中定义的局部系统（第 35-40 行），其中约束通过拉格朗日乘子施加。对于每个补丁 p，单元左端项[p] 和单元右端项[p] 包含局部系统矩阵，并按对应于不同检验变量和试探变量的 2×2 个块进行划分。借助 Gridap 的 LazyArray，我们可以将所有计算推迟到需要局部系统时再进行，因此这些数组从不以完整形式显式存储。该框架还会缓存并复用计算和组装单个局部问题所需的全部内存，从而降低内存占用并提升性能。

结合线性系统的局部组装，我们提供了多种局部映射来求解这些系统。最基本的是 LocalSolveMap，它接受一个整体右端项矩阵和任意数量的左端项数组，并通过 Julia 标准的 LU 求解器返回每个相应局部线性系统的解。不过，要实现利用局部系统结构的专用局部求解器也很简单。在清单 3（第 50–74 行）中，我们展示了如何为混合高阶方法重构算子实现一个自定义局部求解器，该求解器利用局部系统的 2×2 分块结构执行约束求解。在我们的实现中，这是通过一种精确罚函数类方法完成的，详见 [26, 附录 B.2。1], 求解前会先消去拉格朗日乘子。这个自定义求解器定义为一个 Map，其 evaluate! 函数会针对每个补丁调用。因此，输入 lhs 和 rhs 分别对应一对局部系统矩阵，输出则是一对矩阵，其中包含该处重构的系数

补丁。尽管示例实现中未使用，返回缓存函数可用于预分配变量，以便在多次调用 `evaluate!` 时重复使用这些变量。随后，这些缓存会作为 `evaluate!` 的第一个参数传入。关于 Gridap 的`Map`接口的详细说明可参见 [29]。

在清单 3 中，第 43–47 行中延迟组装的右端项和左端项矩阵数组被传入第 77–81 行中的受约束的局部求解器；随后，将重构的系数与重构空间 L 的局部基进行线性组合，从而生成每个单元上的重构基（第 84–94 行）。

补丁组装

```
1  using
2  using
3  using       .
4  using       .
5  using       .
6  using       .
7  using       .
8  using       .
9
10       = 2
11   ( ) =  [1]^2 +  [2]^2
12
13       =            ((0,1,0,1),(6,6))
14    =              (     )
15    =             (ReferenceFE{1},      )
16
17 # Hybrid space
18    =                ( ,Float64,     )
19    =                ( ,Float64,     )
20       =                    (2,(1,1))
21    =                 ([ , ];     )
22
23 # Reconstruction space
24    =                ( ,Float64,     +1)
25    =                ( ,Float64,0)
26    =                 ([ , ];     )
27
28 # Define local problems
29       =         .            (     )
30     =                 (     ,     )
31     =                          (     ,     )
32      =        (  ,2*(     +1))
33      =        (  ,2*(     +1))
34
35    =                  (  )
36    ( ) =    ( )
37   ( , ) =              ( , ,          ( ))
38     (( , ),( , ))   =  ( ( )· ( ) +     +    )
39     (( ,  ),( , )) =  ( (  )· ( ) +       )    +
40                       ((   -  (  ,  ))·  ( ))
41
42 # Local assembly
43       =                (     , , )
44          =                (  ,     , , )
45
46       =                (     , , )
47          =                (  ,     , , )
48
49 # Defining the local solve
50 struct LocalPenaltySolveMap <: Map end
51
52 function       .           (
53   ::LocalPenaltySolveMap,     ,
54 )
55   return         # No cache needed
56 end
57
58 function       .         (
59         , ::LocalPenaltySolveMap,      ,
60 )
61      ,    ,    ,     =          (   )
62      ,    ,    ,     =          (   )
63
64   # App = App + μT * Apλ * Aλp
65   # BpΩ = BpΩ + μT * Apλ * BλΩ
66      =   (   )/    (   )^2
67       (   ,    ,    ,   ,  1)
68       (   ,    ,    ,   ,  1)
69
70        =    (   )
71       =      (    ,   )
72       =      (    ,   )
73   return    ,
74 end
75
76 # Solve local systems
77              =         (
78                        (),          ,
79 )
80               =         (     ,           )
81               =         (    ,           )
82
83 # Build reconstruction fields
84              =         (            ( ))
85               =         (
86                        ,              ,
87 )
88               =         (
89                        ,              ,
90 )
91    =              ()
92    =                (            ,  ,   )
93    =                (            ,  ,   )
94   =                 ([   ,   ])
```

Listing 3. 低层级补丁组装示例，展示了式 (2.4) 中定义的 HHO 重构算子的低层级实现。

4.5. **局部算子**。局部算子封装了通过局部线性求解计算离散投影这一常见模式。如清单 3 所示，局部线性系统的解通常用于定义两个不同有限维空间之间的投影。例如，HHO 重构算子定义了一个椭圆投影算子，将阶数为 $(k,\ l)$ 的局部 HHO 空间投影到单元上次数为 $k+1$ 的多项式空间。尽管这些投影可能各不相同，但它们往往具有共同的结构，因此可以将清单 3 中展示的大部分底层复杂性抽象掉。

为涵盖所有可能性，我们提供了一种新的 LocalOperator 结构。本质上，该对象接受一个用于计算局部线性系统的方案、一个用于求解每个此类系统的 Map，以及一个输出不连续有限元空间，所得局部解属于该空间。

这里有两种主要签名，代码清单 4 对此均进行了展示。第一种签名专用于其线性系统定义在单个几何实体（例如单个单元或面）上的局部算子。它用于（第 2–11 行）创建式（2.6）中定义的 L^2-投影 $\pi^{0,\ k}_{T}$ 。LocalOperator 构造函数接受一个 LocalSolveMap、输出有限元空间，以及分别对应投影左端项和右端项的两个双线性型；在本例中，二者均为 L^2-质量双线性型。第二种签名用于需要在网格片上组装线性系统的局部算子。它用于（第 13–39 行）实现式（2.4）中的混合高阶方法重构算子 $\mathcal{R}_T$ ，从而替代代码清单 3 中所示的大部分手动组装过程。该构造函数接受一个局部惩罚求解映射、一个 PatchTopology、输出空间和输入空间，以及分别对应式（2.4）左端项和右端项的双线性型 lhs 和 rhs。辅助拉格朗日乘子空间 Λ用于在式（2.4）的第二行中施加零均值条件；它与重构空间 L 一同封装到多场有限元空间 W 中（第 33 行）。

一个细微但重要的细节是使用 FESpaceWithoutBCs（第 5 行和第 32 行）。在混合高阶方法的语境中，我们使用局部算子来计算两个有限元空间之间的局部基变换。例如，HHO 重构算子的输入空间是局部 HHO 空间，它结合了单元自由度和面自由度。重构算子是从这个完整的局部 HHO 空间到多项式重构空间的基变换，因此必须作用于每个补丁内的所有 自由度，包括区域边界上的面自由度。然而，默认情况下，Gridap 会在组装过程中消去狄利克雷约束自由度（即边界面自由度）；因此，为重构组装的右端项矩阵中会缺少这些自由度对应的列。 L^2-投影算子的左端项矩阵也会出现同样的问题，因为其输出空间通常是 HHO 骨架空间。FESpaceWithoutBCs 包装器会将狄利克雷自由度重新编号为自由度，同时保留局部排序，确保每个补丁内的所有 HHO 自由度（包括狄利克雷自由度）都参与局部组装，并生成完整的基变换矩阵。

L2 投影与重构算子

```
1
2  function                    ( ,  ,   )
3     ( , ) =              ( , ,          ( ))
4        ( , ) =  (   ( , ))
5       =        .                 ( )
6      =             (
7                  (),    ,     ,     ;
8                =
9    )
10   return
11 end
12
13 function                        ( ,     , ,     )
14     =                 ( ,Float64,     +1)
15     =                 ( ,Float64,0)
16
17      =                   ( ,     )
18      =                           ( ,     )
19
20           = 2*(     +1)
21       =        (  ,       )
22       =        (  ,       )
23
24     =                  (  )
25     ( ) =    ( )
26    ( , ) =              ( , ,          ( ))
27      (( , ),( , ))   =  ( ( )· ( ) +     +    )
28      ((  ,  ),( , )) =  ( (  )· ( ) +      )    +
29                         ((   -  (  ,  ))·  ( ))
30
31         =                    (2,(1,1))
32     =        .                 ( )
33     =                  ([ , ];     )
34     =              (
35                         (),     ,   ,  ,    ,    ;
36                =
37   )
38   return
39 end
```

Listing 4. L^2-投影和 HHO 重构算子使用单元网格片上的 LocalOperator 定义，从而抽象出清单 3 中所示的手动组装过程。该高级接口隐藏了网格片组装和局部求解的底层细节。

4.6. 静态凝聚。 混合方法的一个显著特征是能够通过静态凝聚在局部消除单元未知量，从而得到一个仅涉及骨架（面）自由度的约化全局系统。该过程需要组装耦合单元与面未知量的局部贡献，然后对每个单元执行舒尔补运算，以在全局组装之前消除单元自由度。我们的框架基于第 4.4 节中介绍的网格片组装基础设施，实现了该过程的自动化。

一个关键挑战在于，诸如 $\mathcal{R}_T$ 这样的局部算子会将混合空间 $U^{k,\ l}_{h}(T)$ 映射到定义在单元上的重构空间 $\mathbb{P}^{k+1}(T)$ 。由此得到的重构基与补丁内的所有混合自由度相关联，因此组装时必须将这一点考虑在内。

为了解决这一问题，我们引入了 PatchFESpace 结构。给定一个空间 X 和一个 PatchTopology，PatchFESpace 在清单 5（第 57 行）中通过 Xp = PatchFESpace(X,ptopo) 加以说明，它会创建 X 的一个视图，汇集每个补丁中的所有自由度。在 HHO 的情形下，Xp 会为每个单元 T 汇集该单元及其周围所有面上的局部自由度，从而为组装器提供涉及重构基的各项贡献所需的正确全局自由度索引。

这使得在组装混合原始基与重构基的形式时，自然形成 2×2 分块结构。一方面，一致性项 a_T 仅涉及重构算子，因此可以仅使用 Xp 进行组装。另一方面，稳定化项 S_T同时涉及原始基与重构基。因此，我们需要分解稳定化中出现的面残差。具体而言，对于任意 $u_T \in U^k_{\cdot,h}{}^k$ 以及任意面 $F \in \mathcal{F}_h(T)$，考虑

$$s_{TF}(\underline{u}_T) \doteq (\delta^k_{TF} - \delta^k_T)\underline{u}_T,$$

其中，$\delta^k_T u_T \in \mathbb{P}^k(T)$ 在面表达式中应理解为限制在 F上。利用 δ^k_T和δ^k_{TF}的定义，我们可以在每个面 F上写为:

$$s_{TF}(\underline{u}_T) = S_1(\underline{u}_T) + S_2(\mathcal{R}_T\underline{u}_T),$$

with

$$S_1(\underline{u}_T) \doteq (u_T)|_F - u_F, \quad \text{and} \quad S_2(\mathcal{R}_T\underline{u}_T) \doteq \pi^{0,k}_F\big((\mathcal{R}_T\underline{u}_T)|_F\big) - \big(\pi^{0,k}_T(\mathcal{R}_T\underline{u}_T)\big)|_F.$$

项 S_1 和 S_2 分别对应原始自由度和重构自由度。于是，稳定化可以重写为

$$S_T(\underline{u}_T, \underline{v}_T) = h_T^{-1} \sum_{F\in\partial T} \int_F \big(S_1(\underline{u}_T) + S_2(\mathcal{R}_T\underline{u}_T)\big) \cdot \big(S_1(\underline{v}_T) + S_2(\mathcal{R}_T\underline{v}_T)\big) \tag{4.5}$$

$$= h_T^{-1} \sum_{F\in\partial T} \int_F S_1(\underline{u}_T) \cdot S_1(\underline{v}_T) + h_T^{-1} \sum_{F\in\partial T} \int_F S_1(\underline{u}_T) \cdot S_2(\mathcal{R}_T\underline{v}_T) \tag{4.6}$$

$$+ h_T^{-1} \sum_{F\in\partial T} \int_F S_2(\mathcal{R}_T\underline{u}_T) \cdot S_1(\underline{v}_T) + h_T^{-1} \sum_{F\in\partial T} \int_F S_2(\mathcal{R}_T\underline{u}_T) \cdot S_2(\mathcal{R}_T\underline{v}_T) \tag{4.7}$$

清单 5 中的第 66–72 行将这四个分块传递给 collect and merge_cell matrix and vector，从而将它们组装为局部 2×2 分块矩阵，使单元与面未知量相互耦合。每个分块均使用适当的空间对进行组装：对于仅涉及原始自由度的贡献，使用 (X, X)；对于试函数侧和检验侧均涉及重构的项，使用 (Xp, Xp)；对于交叉项，使用混合空间对 (X, Xp) 和 (Xp, X)。PatchAssembler 根据各空间提供的自由度索引，确保将各项贡献放置在全局矩阵中的正确位置。

清单 5 展示了底层机制和高级接口。第 75–87 行展示了一个自定义的 StaticCondensationMap，它显式执行舒尔补：提取 2×2 矩阵块 K_{ii} (单元-单元)、 K_{bb} (面-面)、K_{ib} (单元-面) 和 K_{bi} (面-单元)，以及右端项分块 b_i 和 b_b。随后，它通过计算来消去内部自由度

$$\tilde{K}_{bb} = K_{bb} - K_{bi}K_{ii}^{-1}K_{ib} \quad \text{and} \quad \tilde{b}_b = b_b - K_{bi}K_{ii}^{-1}b_i.$$

第 89–97 行则手动组装凝聚骨架系统。对于典型用例，高级 StaticCondensationOperator（第 100–103 行）会自动完成整个过程，使用户能够以声明式方式表达该方法，而无需管理底层组装细节。

4.7.最终驱动程序。 本节介绍的抽象——多面体表示、间断多项式空间、基于片区的组装、局部算子和自动静态凝聚——共同构成了一个用于实现混合方法的统一框架。借助 Gridap 的惰性求值和多重分派，这些工具兼具高级表达能力与计算效率。代码清单 6 展示了一个采用 HHO 离散求解泊松问题的完整实现，其中使用了本节介绍的所有构件。

在下一节中，我们将演示如何修改这些组件，以便为日益复杂的问题简洁地实现间断伽辽金、混合间断伽辽金和 HHO 方法。

静态凝聚

```
1  using
2  using
3  using       .
4  using       .
5  using       .
6  using       .
7  using       .
8
9        = 2
10        ( ) =  [1]^2 +  [2]^2
11  ( ) = - (       )( )
12
13       =            ((0,1,0,1),(6,6))
14   =             (     )
15   =             (ReferenceFE{1},      )
16
17 # Hybrid space
18   =                ( ,Float64,     )
19   =                ( ,Float64,     )
20       =                    (2,(1,1))
21   =                 ([ , ];     )
22
23 # Define local problems
24       =         .            (     )
25    =                  (     ,     )
26    =                         (     ,     )
27     =        (  ,2*(     +1))
28     =        (  ,2*(     +1))
29
30 # Operators
31    =          (    .(                ( )),    )
32    =                        ( ,     , ,     )
33    =                    ( ,   ,    )
34    =                    ( ,   ,    )
35
36 # Problem definition
37 function  ((   ,   ),(   ,   ))
38       =  (   ) +  (   )
39       =  (   ) +  (   )
40   return  (    ·    )
41 end
42
43  ((  ,  )) =  (    )
44  ( , ) =  ((1/  )*(   ))
45
46 function   ((  ,  ))
47   return   (  ) -
48 end
49 function   ((   ,   ))
50        ,       =   (   ),   (   )
51        ,       =   (   ),   (   )
52   return (      -      ) + (      -      )
53 end
54
55 # Local assembly
56       =                (     , , )
57    =         .            ( ,     )
58  ,    =                 ( ),              ( );
59
60   ,    =  ( ),  ( )
61    ,     =   ( ),   ( )
62    ,     =   (  ),   (  )
63   =                   ()
64
65      =                                        (
66        ,
67   (  ,   ,  (   ,   ),           ( ),      ( ) ),
68   (  ,   ,  (   ,   ),              ,      ( ) ),
69   (  ,   ,  (   ,   ),              ,      (  )),
70   (  ,   ,  (  ,  ) +  (   ,   ),   ,      (  )),
71 )
72              =                           (     ,    )
73
74 # Manual static condensation
75 struct StaticCondensationMap <: Map end
76
77 function       .         (
78        ,::StaticCondensationMap,
79 )
80      ,     =
81      ,    ,    ,     =          (   )
82     ,    =          (   )
83     =    (   )
84        ( ,  )
85        ( ,   )
86       (  ,   ,  ,-1,1)
87       (   ,   ,   ,-1,1)
88   return    ,
89 end
90
91                =                      ( , )
92                 =         (
93                       (),              )
94      =         .                      ( ,     )
95      = (([              , ], [   ,], [   ,]),
96         ([],[],[]),  ([],[]))
97  ,   =                          (              ,     )
98
99 # Static condensation high-level API
100    =                          (
101    ,     ,
102 )
103  ,     =      (  )
```

Listing 5. 静态凝聚示例展示了 2×2 块组装结构。第 66–72 行演示了如何将贡献拆分到原始自由度 (X) 和片自由度 (Xp) 中。代码清单展示了底层机制（第 75–97 行）以及高层接口（第 100–103 行）。

5. Examples

5.1. 第一个示例：泊松问题。作为第一个示例，我们考虑泊松问题 (2.2)。在以下各小节中，我们将演示如何使用第 4 节介绍的框架实现三种复杂程度递增的方法。首先介绍间断伽辽金，以展示分片多项式空间的基本用法（第 5.1.1 节）。接着介绍混合间断伽辽金，它引入了混合空间和静态凝聚（第 5.1.2 节）。最后展示 HHO 的结果，该方法还使用了局部投影算子（第 5.1.3 节）。

在这三个示例中，我们将在区域 $\Omega = [0,1]^2$ 中使用制造解方法，并考虑精确解 $u^* = x_1^2 + x_2^2$。我们取 $f = -\Delta u^*$ 和 $g = u^*|_{\partial\Omega}$。我们将使用图 3 所示的代码，对由含有 $n\times n$ 个单元的单纯形化笛卡尔网格生成的沃罗诺伊剖分 $\mathcal{M}_h$ 对区域 Ω 进行网格划分。

5.1.1. 泊松问题的间断伽辽金离散。 清单 7 展示了泊松问题间断伽辽金离散的实现 [14]。给定多项式次数 $k > 0$，我们将空间取为

示例：HHO 泊松

```
1  using
2  using
3  using       .        ,        .
4  using       .          ,        .
5  using       .            ,        .
6
7          ( ) =  [1] +  [2]
8   ( ) = - (        )( )
9
10        =             ((0,1,0,1),(6,6))
11   =              (ReferenceFE{2},       )
12   =              (ReferenceFE{1},       )
13
14        = 1
15
16        =              (     )
17    =                  (     ,     )
18    =                          (     ,     )
19
20          = 2*(     +1)
21     =        (  ,        )
22     =        (  ,        )
23
24   =                ( , Float64,       )
25   =                (
26    , Float64,       ;              =
27 )
28   =             ( ,        )
29
30        =                    (2,(1,1))
31    =                  ([ ,  ];     =     )
32    =                  ([ ,  ];     =     )
33
34    =          (    .(                ( )),    )
35    =                        ( ,     , ,     )
36    =                    ( ,   ,    )
37    =                    ( ,   ,    )
38
39 function  ((   ,   ),(   ,   ))
40        =  (   ) +  (   )
41        =  (   ) +  (   )
42   return  (    ·    )
43 end
44
45  ((  ,  )) =  (    )
46  ( , ) =  ((1/  )*(   ))
47
48 function   ((  ,  ))
49   return   (  ) -
50 end
51 function   ((   ,   ))
52        ,       =   (   ),   (   )
53        ,       =   (   ),   (   )
54   return (      -      ) + (      -      )
55 end
56
57 function         (     ,     , , )
58      =         .            ( ,     )
59    ,   =                  ( ),            ( );
60
61     ,    =  ( ),   ( )
62      ,     =   ( ),   ( )
63      ,     =   ( ),    (  )
64     =                   ()
65
66        =                                          (
67          ,
68     (  ,   ,  (   ,   ),          ( ),      ( ) ),
69     (  ,   ,  (   ,   ),              ,     ( ) ),
70     (  ,   ,  (   ,   ),              ,     (  )),
71     (  ,   ,  (  ,  ) +  (   ,   ),   ,     (  )),
72   )
73   return                          (     ,    )
74 end
75
76        =              (     , , )
77    =                          (
78    ,     ,        (     ,     , , )
79 )
80
81    =      (  )
82   ,    =
83
84     =  (  ) -
85        =     (   (  (   *   )   ))
86        =        +     (   (  ( (  ) ·  (  ))   ))
```

Listing 6. 泊松问题的 HHO 离散（第 5.1.3 节）。需要图 3 中的代码和代码清单 4 中的代码。

未知量所在的分片多项式空间 $V_h = \mathbb{P}^k(\mathcal{M}_h)$ （第 13 行）。给定一个内部面 $F \in \mathcal{F}_h^{\text{in}}$及其两个相邻单元 T，$T' \in \mathcal{T}_h$，我们为标量函数 v和张量函数 $\boldsymbol{\sigma}$ 定义跳跃算子和平均算子如下

$$\{\{\boldsymbol{\sigma}\}\} = \frac{1}{2}\left(\boldsymbol{\sigma}_{|T} + \boldsymbol{\sigma}_{|T'}\right), \quad [\![v]\!] = v_{|T}\boldsymbol{n}_{FT} + v_{|T'}\boldsymbol{n}_{FT'},$$

其中，$\boldsymbol{n}_{FT}$和 $\boldsymbol{n}_{FT'}$是 F上的单位法向量，分别从 T和 T'指向外侧（第 24–25 行）。对于与单元 $T \in \mathcal{T}_h$相邻的边界面 $F \in \mathcal{F}_h^{\text{bd}}$，定义

$$\{\{\boldsymbol{\sigma}\}\} = \boldsymbol{\sigma}_{|T}, \quad [\![v]\!] = v_{|T}\boldsymbol{n}_{FT}.$$

离散问题（第 30–45 行）如下：求 $u_h \in V_h$ 使得

$$a_h(u_h, v_h) = l_h(v_h) \quad \forall v_h \in V_h,$$

其中

$$a_h(u,v) = \sum_{T\in\mathcal{T}_h}\int_T \nabla u\cdot\nabla v\,dx + \sum_{F\in\mathcal{F}_h}\int_F\left(\frac{\gamma}{h_F}[\![v]\!][\![u]\!] - \{\{\nabla u\}\}\cdot[\![v]\!] - \{\{\nabla v\}\}\cdot[\![u]\!]\right)ds$$

$$l_h(v) = \sum_{T\in\mathcal{T}_h}\int_T fv\,dx + \sum_{F\in\mathcal{F}_h^{\text{bd}}}\int_F\left(\frac{\gamma}{h_F}gv - (\nabla v\cdot\boldsymbol{n}_F)g\right)ds,$$

其中，$\gamma > 0$ 是罚参数（第 29 行），h_F 是面 F 的直径（第 26–27 行）。该方法的收敛结果见图 4。

示例：DG 泊松

```
1
2  using
3  using
4  using       .
5  using       .
6
7   ( ) =  [1]^2 +  [2]
8   ( ) = -  ( )( )
9
10       =             ((0,1,0,1),(6,6))
11
12       = 2
13   =                 (     ,Float64,     )
14
15   =         (     )
16   =         (     )
17   =         (     )
18
19         = 2*
20    =        ( ,       )
21    =        ( ,       )
22    =        ( ,       )
23
24    =                  ( )
25    =                  ( )
26    =          (                ( ),   )
27    =          (                ( ),   )
28
29    = 10 # penalty parameter
30   ( , ) =
31     (  ( )· ( ) )*   +
32     ( ( /  )* *   -  *(    ( ))
33       - (    ( ))*
34      )*   +
35     ( ( /  )*    ( *  )·    ( *  )
36       -     ( *  )·    ( ( ))
37       -     ( ( ))·    ( *  )
38      )*
39
40   ( ) =
41     (  *  )*   +
42     ( ( /  )* *  - (    ( ))*  )*
43
44     =                 ( , , , )
45     =      (  )
46
47     =   -
48         =     (   ( (     )  ))
49         =     (   ( (      +  (  )⊙ (  ))  ))
50
51          (
52     ,          ,
53              = [
54          =>   ,      =>   ,     =>
55    ]
56  )
```

LISTING 7. 泊松问题的间断伽辽金离散（第 5.1.1 节）。需要使用图 3 中的代码。

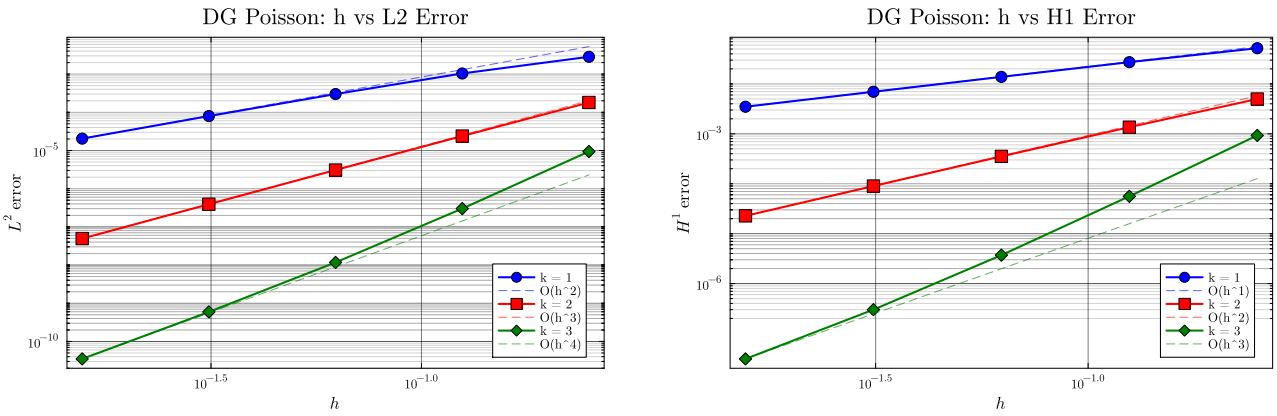

FIGURE 4. 泊松问题的 DG 方法收敛性： L^2-误差（左）和 H^1-误差（右）随网格尺寸 h 的变化，针对多项式次数 $k = 1,2,3$。参考线显示了相应范数下的预期收敛率。结果由代码清单 7 中的驱动程序生成，使用的是按照图 3 所述方式构造的 Voronoi 网格。

5.1.2. 泊松问题的 *HDG* 离散。 HDG 方法 [15]通过引入通量 $\boldsymbol{p} = -\nabla u$ 作为辅助变量，将泊松问题改写为一阶系统。于是问题可表述为：求 $(u, \boldsymbol{p})$ 使得

$$\boldsymbol{p} + \nabla u = \boldsymbol{0} \quad \text{in} \quad \Omega, \quad \nabla \cdot \boldsymbol{p} = f \quad \text{in} \quad \Omega, \quad u = g \quad \text{on} \quad \partial\Omega.$$

代码清单 8 展示了此示例的实现。给定多项式次数 $k \geqslant 0$，我们定义三个未知量空间：单元未知量空间 $V_h = \mathbb{P}^{k+1}(\mathcal{T}_h)$ （第 17 行）、通量空间$Q_h = \left[\mathbb{P}^k\right]^d(\mathcal{T}_h)$ （第 18–20 行）和迹空间 $M_h = \mathbb{P}^k(\mathcal{F}_h)$ （第 21–24 行）。因此，我们将混合空间定义为 $U_h = Q_h \times V_h \times M_h$（第 26–27 行），其解的形式为 $u_h = (\boldsymbol{p}_h,\ u_h,\ s_h) \in U_h$。对于每个单元 $T \in \mathcal{T}_h$，我们用 $U_h(T) = Q_h|_T \times V_h|_T \times M_h|_T = \left[\mathbb{P}^k(T)\right]^d \times \mathbb{P}^{k+1}(T) \times \mathbb{P}^k(\mathcal{F}_h(T))$ 表示 T上的局部未知量空间，其中包含形式为 $u_T = (\boldsymbol{p}_T,\ u_T,\ s_T) \in U_h(T)$的局部解。请注意，BlockMultiFieldStyle(2,(2,1))（第 26 行）将三字段空间组织为双块系统 $(Q_h \times V_h,\ M_h)$，其中第一个块包含所有将进行静态凝聚的变量。

离散问题如下（第 37–47 行）：求 $u_h \in U_h$，使得 $\forall \underline{v}_h \in U_h$ 满足

$$a_h(\underline{u}_h, \underline{v}_h) = l_h(\underline{v}_h),$$

$$a_h(\underline{u}_h, \underline{v}_h) = \sum_{T \in \mathcal{T}_h} a_T(\underline{u}_T, \underline{v}_T) + S_T(\underline{u}_T, \underline{v}_T), \quad l_h(\underline{v}_h) = \sum_{T \in \mathcal{T}_h} l_T(\underline{v}_T),$$

其中

$$a_T(\underline{u}_T, \underline{v}_T) = \int_T (\boldsymbol{p}_T \cdot \boldsymbol{q}_T - u_T \nabla \cdot \boldsymbol{q}_T - \boldsymbol{p}_T \cdot \nabla v_T) + \sum_{F \in \mathcal{F}_h(T)} \int_F (s_F \boldsymbol{q}_T \cdot \boldsymbol{n}_{TF} + \boldsymbol{p}_T \cdot \boldsymbol{n}_{TF} m_F),$$

$$S_T(\underline{u}_T, \underline{v}_T) = \tau \sum_{F \in \mathcal{F}_h(T)} \int_F \left((\pi_F^{0,k} u_T - s_F)(\pi_F^{0,k} v_T + m_F) \right), \quad \text{and} \quad l_T(\underline{v}_T) = \int_T f v_T \, dx.$$

注意，稳定化项 $S_T(u_T, v_T)$ 并非标准的 HDG 稳定化，而是经过轻微修改的版本，可在多面体网格上保持稳健 [20]。该方法的收敛结果见图 5。

示例：混合差分伽辽金泊松方法

```
 1 using
 2 using       .        ,       .
 3 using       .          ,       .
 4 using       .        ,       .
 5 using       .            ,       .
 6 using       .
 7
 8        ( ) =  [1] +  [2]
 9  ( ) = - (       )( )
10  ( ) = (  ·  )( )
11
12       =             ((0,1,0,1),(6,6))
13   =              (ReferenceFE{2},      )
14   =              (ReferenceFE{1},      )
15
16       = 1
17   =                ( , Float64,      +1)
18   =                (
19    , VectorValue{2, Float64},
20 )
21   =                (
22    , Float64,      ;              =
23 )
24   =             ( ,        )
25
26       =                    (2,(2,1))
27   =                 ([ ,  ,  ];     )
28
29       =              (     )
30   =                  (     ,     )
31   =                          (     ,     )
32
33        = 2*(     +1)
34    =        (  ,       )
35    =        (  ,       )
36
37  = 1 /          (                (  ),    )
38  =                  (  )
39  ( ) =
40 ( ) =              ( ,  ,           ( ))
41   =                   ( ,   ,    )
42
43 ((  ,  ,  ),(  ,  ,  )) =
44   (      -   *(    ) -     (  ))    +
45   ((  (  ) +  *(  (  ) -   ))*( (  ) +   ))    +
46   (  *  (  ))
47 ((  ,  ,  )) =  ( *  )*
48
49   =                          (     , , , )
50  ,   ,    =      (  )
51
52   =    -
53   =        ( ,       )
54       =     (   ( (     )*  ))
```

Listing 8. 泊松问题的 HDG 离散（第 5.1.2 节）。需要使用图 3 中的代码和代码清单 4 中的代码。

5.1.3. 泊松问题的 *HHO* 离散。 第 2 节详细介绍了用于泊松问题的 HHO 方法。代码清单 6 展示了使用第 4 节介绍的框架实现该方法的完整代码。重构算子的实现见代码清单 4。该方法的收敛结果见图 6。

5.2. **更复杂的示例**。 现在，我们讨论文献中一些更复杂的标准示例，以展示所提出框架的灵活性，以及将其扩展到新公式的便捷性。

5.2.1. 线弹性问题中的混合高阶方法。 线弹性是一个向量值问题，其核包含所谓的刚体模态。线弹性的混合阶 HHO 公式 [19] 与上文介绍的泊松问题的公式非常相似，主要变化在于修改局部重构算子以考虑刚体运动，并使用向量值空间。我们将紧密遵循 [26, 第 7]章中介绍的公式。代码清单 9 展示了该示例的实现。为给前一个示例增加一些变化，我们将考虑混合阶 HHO 公式。给定多项式次数 $k \geqslant 0$，我们将混合空间（第 23–31 行）定义为

$$\underline{\boldsymbol{U}}_h^{k+1,k} = \left[\mathbb{P}^{k+1}\right]^d (\mathcal{T}_h) \times \left[\mathbb{P}^k\right]^d (\mathcal{F}_h). \tag{5.1}$$

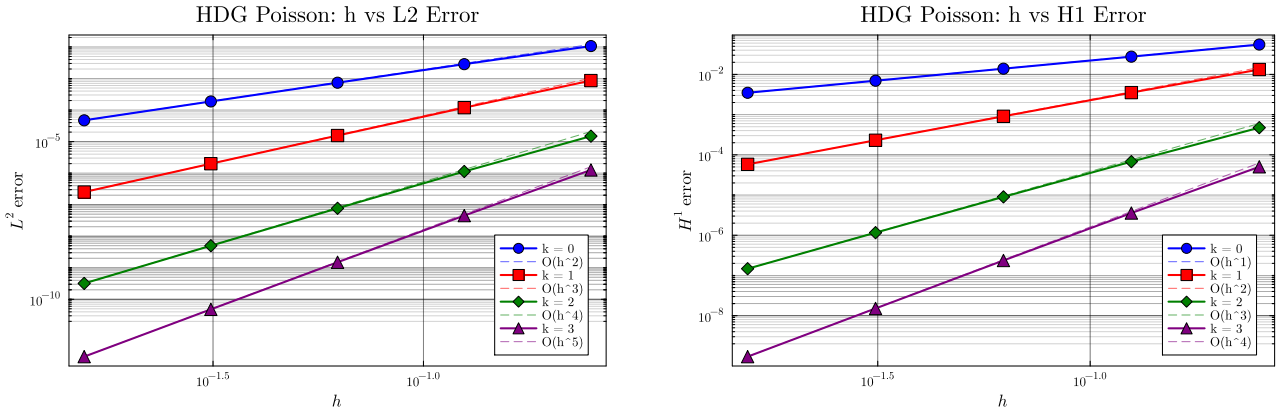

FIGURE 5. 混合间断伽辽金方法求解泊松问题的收敛性： L^2 误差（左）和 H^1误差（右）随网格尺寸 h 的变化，多项式次数为 $k = 0$，1，2，3。参考线显示理论收敛率： $O(h^{k+2})$ 对应 L^2， $O(h^{k+1})$ 对应 H^1。结果由代码清单 8 中的驱动程序生成，所用沃罗诺伊网格按照图 3 所述方法构造。

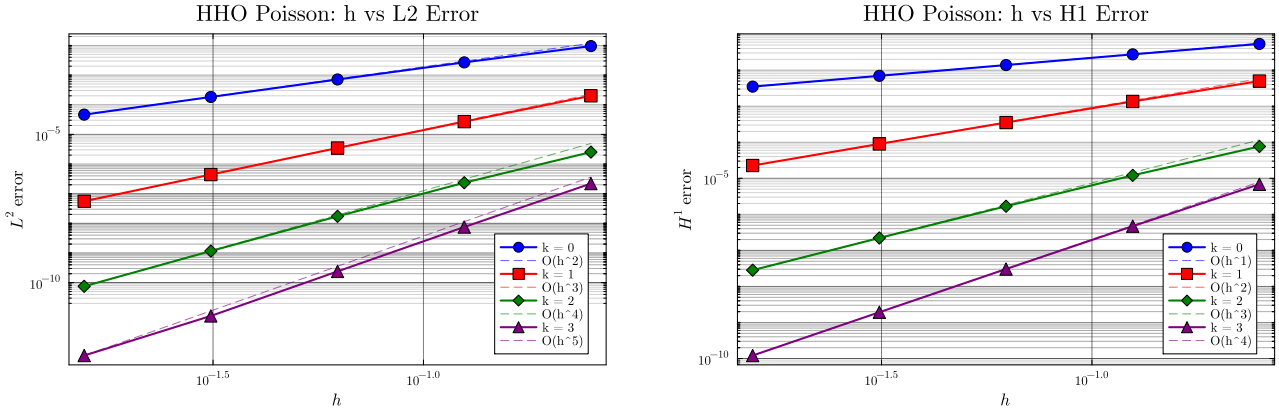

FIGURE 6. 混合高阶方法求解泊松问题的收敛性： L^2 误差（左）和 H^1误差（右）随网格尺寸 h 的变化，多项式次数为 $k = 0$，1，2，3。参考线显示理论收敛率： $O(h^{k+2})$ 对应 L^2， $O(h^{k+1})$ 对应 H^1，体现了超收敛性。结果由代码清单 6 中的驱动程序生成，所用沃罗诺伊网格按照图 3 所述方法构造。

局部重构算子 $\mathcal{R}_T^{\mathrm{sym}}$： $\boldsymbol{U}^{k+1}{}_{,h}{}^{k}(T) \to \underline{\left[\mathbb{P}^{k+1}(T)\right]^d}$现在由下式唯一确定

$$(\nabla_s \mathcal{R}_T^{\mathrm{sym}} \underline{v}_T, \nabla_s w)_T = (\nabla_s v_T, \nabla_s w)_T + \sum_{F \in \mathcal{F}_h(T)} (v_F - v_T, \nabla_s w \cdot \boldsymbol{n}_{TF})_F \quad \forall w \in \left[\mathbb{P}^{k+1}\right]^d(T), \tag{5.2}$$

$$(\mathcal{R}_T^{\mathrm{sym}} \underline{v}_T, 1)_T = (v_T, 1)_T \tag{5.3}$$

$$(\nabla_s \mathcal{R}_T^{\mathrm{sym}} \underline{v}_T, 1)_T = \sum_{F \in \mathcal{F}_h(T)} \frac{1}{2} (\boldsymbol{n}_{TF} \otimes v_F - v_F \otimes \boldsymbol{n}_{TF}, 1)_F \tag{5.4}$$

请注意，这个新的重构算子求解的是向量泊松问题的一种修改形式（见（2.4））：我们将梯度替换为对称梯度 ∇_s，并将刚体运动加入核空间。该算子的实现见列表 10。约束通过拉格朗日乘子施加，使用单元分片常向量值空间，每个约束对应一个分量： $\Lambda 1$ 施加平均值约束（逐分量包含 d 个方程），而 $\Lambda 2$ 施加刚体模态约束（包含 $d(d+1)/2$ 个方程），使用内联函数 α（逐分量缩并）和 $\beta(\boldsymbol{n}, v) = (1/2)(\boldsymbol{n} \otimes v - v \otimes \boldsymbol{n})$。

混合阶 HHO 用于弹性问题的优势在于，我们可以用更简单的梯度重构算子 $\mathcal{G}_T$ 替换 $\mathcal{R}_T^{\mathrm{sym}}$： $\boldsymbol{U}^{k+1}{}_{,h}{}^{k}(T) \to \left[\mathbb{P}^{k+1}(T)\right]^{d\times d}$ 对称 映射到 __

对称二阶张量子空间，其定义为

$$(\mathcal{G}_T\underline{v}_T,\boldsymbol{\tau})_T=(\nabla_s v_T,\boldsymbol{\tau})_T+\sum_{F\in\mathcal{F}_h(T)}(v_F-v_T,\boldsymbol{\tau}\cdot\boldsymbol{n}_{TF})_F\quad\forall\boldsymbol{\tau}\in\left[\mathbb{P}^{k+1}(T)\right]^{d\times d}_{\mathrm{sym}}.\tag{5.5}$$

该算子的实现也可以在清单 10 中找到。

回到代码清单 9，离散问题可写为（第 50–72 行）：求解 $u_h\in\boldsymbol{U}^{k+1}_{,h}\ \ ^{k}_{\_}$ 使得$\underline{a}_h(u_h,v_h)_=\underline{l}_h(v_h)\quad\forall\underline{v}_h\in\boldsymbol{U}^{k+1}_{_,h_}\ ^{k}$。所涉及的局部形式定义如下：

$$a_T(\underline{u}_T,\underline{v}_T)=(\mathcal{G}_T\underline{u}_T,\boldsymbol{C}:\mathcal{G}_T\underline{v}_T)_T\,,$$

$$S_T(\underline{u}_T,\underline{v}_T)=\sum_{F\in\mathcal{F}_h(T)}h_F^{-1}\int_F\Big((\pi_F^{0,k}u_T-u_F)\cdot(\pi_F^{0,k}v_T-v_F)\Big),\quad\text{and}\quad l_T(\underline{v}_T)=\int_T f v_T\,dx,$$

其中，$\boldsymbol{C}$ 是弹性张量，我们采用由 Lam´e 参数 λ 和 μ 定义的各向同性弹性张量（第 9–12 行）。该方法的收敛结果见图 7。

示例：混合阶 HHO 弹性

```
1  using
2  using       .        ,        .
3  using       .          ,        .
4  using       .            ,        .
5  using       .
6
7  using       .      :
8
9   ,    = 1.0,  1.0
10   =     (SymTensorValue{2,Float64})
11   =     (SymFourthOrderTensorValue{2,Float64})
12   =   *       +   *    # Isotropic tensor
13
14  ( ) =      (   (2 * [1]),   (2 * [2]))
15  ( ) =    ( ( )( ))
16  ( ) = -(   )( )
17
18       =             ((0,1,0,1),(6,6))
19   =              (     )
20   =              (ReferenceFE{2},      )
21   =              (ReferenceFE{1},      )
22
23       = 2
24   =                  (
25    , VectorValue{D,Float64},      +1
26 )
27   =                  (
28    , VectorValue{D,Float64},       ;
29                  =
30 )
31   =             ( , )
32
33       =              (     )
34    =                   (     ,     )
35    =                         (     ,     )
36
37         = 2*(     +1)
38     =         (  ,       )
39     =         (  ,       )
40
41       =                    (2,(1,1))
42     =                 ([ ,  ];     =     )
43     =                 ([ ,  ];     =     )
44
45     =                   ( ,   ,    )
46     =                          ( ,     , ,     )
47     =                      ( ,     , ,     )
48     =           (    .(                ( )),    )
49
50  function  ( , )
51        ,      =  ( )
52        ,      =  ( )
53    return  ((    +    ) ⊙   ⊙ (    +    ))
54  end
55
56  function  ( , )
57     (( ,   )) =   ( ) -
58    return  ((1/  ) * (  ( )·  ( )))
59  end
60
61   (( ,  )) =  (    )
62
63  function          (     ,     , , )
64       =          .              (,      )
65    ,    =                   ( ),              ( )
66         =                                        (
67           ,
68      (  ,   ,  ( , ),                    (),       ( )),
69      ( ,  ,  ( , ),   ( ),      ( ))
70    )
71    return                              (     ,    )
72  end
73
74        =              (     , , )
75     =                        (
76   ,      ,          (     ,     , , )
77 )
78     =      (  )
79
80   ,    =
81    =  (  )
82
83     =         ( ,         )
84          =     (   ( ((   -  )·(    -  ))*  ))
```

Listing 9. 线弹性问题的混合阶HHO离散（第 5.2.1 节）。需要使用图 3 中的代码和清单 10 中的代码。

5.2.2. 用于不可压缩斯托克斯问题的混合高阶方法。另一个典型的向量值问题是不可压缩斯托克斯问题 [1], ，它描述了低雷诺数下不可压缩黏性流体的流动。我们将紧密遵循 [26, 第 8]章中介绍的公式。列表 11 展示了此示例的实现。

对于速度场，我们采用 $\boldsymbol{U}^{k+1}_{,\ h\_\_}\ ^{k}$ 如（5.1）（第 16–23 行）所示；对于压力场，我们采用分片多项式空间 $\mathbb{P}^k(\mathcal{T}_h)$，并将其分解为平均值子空间和零均值子空间。由于直接在压力空间上施加全局零均值约束不可扩展，因此我们改为

对称重构与梯度算子

```
1  function                                  (
2    ,     , ,
3  )
4      =                  (
5      ,VectorValue{2,Float64},      +1
6    )
7      =                  (
8      ,VectorValue{2,Float64},0
9    )
10     =                  (
11     , VectorValue{3,Float64}, 0
12   )
13
14     =                    ( ,      )
15     =                              ( ,      )
16
17           = 2*(      +1)
18      =         (   ,        )
19      =         (   ,        )
20
21    =                    (   )
22    ( ) =     ( )
23   ( , ) =                ( , ,              ( ))
24   ( ) =                            ( )
25   ( , ) =  [1,1]* [1] +  [1,2]* [2] +  [2,2]* [3]
26   ( , ) = -0.5*(     -     )
27     (( ,   ,   ),( ,   ,   )) =
28     ( ( )⊙ ( ) +       +
29      +   ( ( ),   ) +   ( ( ),   ))
30     ((   ,   ),( ,   ,   )) =
31     (( (   )⊙ ( )) + (      ))     +
32     ((    -   (   ,   ))·   ( )
33      +   (   (   , ),   ))
34
35        =                        (2,(1,2))
36    =          .                  ( )
37    =                  ([ ,   ,   ];      )
38    =              (
39                        (),       ,    ,   ,     ,     ;
40              =
41   )
42   return
43 end
44 function                    ( ,      , ,      )
45     =              ( )
46     =                  (
47     ,SymTensorValue{D,Float64},
48   )
49
50     =                    ( ,      )
51     =                              ( ,      )
52
53           = 2*(      +1)
54      =         (   ,        )
55      =         (   ,        )
56
57     =                    (   )
58    ( , ) =                ( , ,              ( ))
59      (( ,),( ,))     =  (    )
60      ((   ,   ),( ,)) =  ( (   )⊙ )     +
61                          ((    -   (   ,   )) · (    ))
62
63         =                        ()
64     =          .                  ( )
65     =                  ([ ];      )
66     =              (
67                  (),       ,    ,   ,     ,     ;
68              =
69   )
70   return
71 end
72
73
74
75
76
77
78
79
80
81
82
```

Listing 10. 线弹性问题（第 5.2.1 节）的混合高阶方法局部算子：对称重构算子和梯度算子。

考虑逐单元零均值压力空间，并通过拉格朗日乘子施加全局零均值约束。这得到以下压力空间（第 25–30 行）：

$$P_h = \hat{P}_h \oplus \bar{P}_h \oplus \Lambda\,, \quad \hat{P}_h = \left\{ q_h \in \mathbb{P}^k(\mathcal{T}_h) : (q_h, 1)_T = 0\ , \forall T \in \mathcal{T}_h \right\}\,, \quad \bar{P}_h = \mathbb{P}^0(\mathcal{T}_h)\,, \quad \Lambda = \mathbb{P}^0(\Omega). \tag{5.6}$$

$\hat{}$子空间 P_h 可与单元速度未知量一起进行静态凝聚，在全局系统中仅保留$\bar{}$均值压力未知量 P_h 和拉格朗日乘子 Λ 。因此，我们使用 BlockMultiFieldStyle(2,(2,3))（第 32–34 行）将五场空间组织成双块系统 $(\boldsymbol{U}_h \times \hat{}P_h,\ U_h^\partial \times \bar{P}_h \times \Lambda)$，其中第一块包含所有将进行静态凝聚的变量。

为了重构速度场，我们将使用式（2.4）中引入的重构算子的向量值版本。给定 $v_T \in \boldsymbol{U}_h(T)$，局部速度重构算子 $\boldsymbol{\mathcal{R}}_T$: $\boldsymbol{U}^{k+1}{}_h{}^k(T) \to \left[\mathbb{P}^{k+1}(T)\right]^d$ 由下式唯一确定：

$$(\nabla \boldsymbol{\mathcal{R}}_T \underline{v}_T, \nabla \boldsymbol{w})_T = -(v_T, \Delta \boldsymbol{w})_T + \sum_{F \in \mathcal{F}_h(T)} (v_F - v_T, \nabla \boldsymbol{w} \cdot \boldsymbol{n}_{TF})_F \quad \forall \boldsymbol{w} \in \left[\mathbb{P}^{k+1}(T)\right]^d, \tag{5.7}$$

$$(\boldsymbol{\mathcal{R}}_T \underline{v}_T, 1)_T = (v_T, 1)_T \tag{5.8}$$

为了重构散度，我们将引入一个新的局部算子 $\mathcal{D}_T$: $\boldsymbol{U}^{k+1}{}_h{}^k(T) \to \mathbb{P}^k(T)$ 定义如下

$$(\mathcal{D}_T \underline{v}_T, q)_T = (\nabla \cdot v_T, q)_T + \sum_{F \in \mathcal{F}_h(T)} ((v_F - v_T) \cdot \boldsymbol{n}_{TF}, q)_F \quad \forall q \in \mathbb{P}^k(T) \tag{5.9}$$

这些算子的实现见清单 12。一个细节值得注意：在清单 11 中使用这些算子时，算子的输入空间是 $\boldsymbol{U}_h$（即第 45–47 行中的 YR），而问题的解空间是 $\boldsymbol{U}_h \times P_h$。因此，我们需要将五场变量转换到速度重构算子的输入空间 $\boldsymbol{U}_h$（然后再将所得的基转换回原始空间）。为此，我们在函数 adapt ids（第 48–54 行）中封装这些算子（第 55–60 行）。或者，也可以仿照清单 3 创建一个新的局部映射，使其能够处理这五个场。

回到清单 11，离散问题如下（第 65–99 行）：求 $(u_h, p_h) \in \boldsymbol{U}_h^{k+1} \times P_h^k$，使得 $a_h((u_h; p_h); (v_h, q_h)) = l_h(v_h) \quad \forall (v_h, q_h) \in \boldsymbol{U}_h^{k+1} \times P_h^k$，其中局部形式定义如下：

$$a_T(\underline{u}_T, p_T; \underline{v}_T, q_T) = \int_T (\nabla \boldsymbol{\mathcal{R}}_T \underline{u}_T : \nabla \boldsymbol{\mathcal{R}}_T \underline{v}_T - p_T \mathcal{D}_T \underline{v}_T - q_T \mathcal{D}_T \underline{u}_T),$$

$$S_T(\underline{u}_T, \underline{v}_T) = \sum_{F \in \mathcal{F}_h(T)} h_F^{-1} \int_F \left((\pi_F^{0,k} u_T - u_F) \cdot (\pi_F^{0,k} v_T - v_F) \right), \quad \text{and} \quad l_T(\underline{v}_T) = \int_T \boldsymbol{f} \cdot v_T \, dx.$$

该方法的收敛结果见图 7。

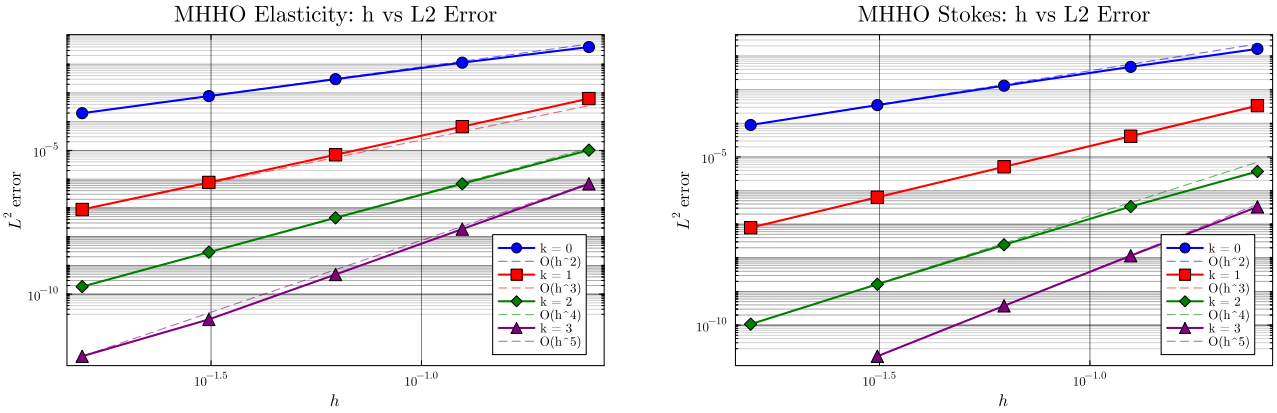

FIGURE 7. 混合阶 HHO 方法的收敛性：L^2-误差与网格尺寸 h的关系，其中多项式次数为 $k = 0$、1、2、3。线弹性（左）和斯托克斯（右）。参考线显示预期的收敛速率 $O(h^{k+2})$。结果基于按照图 3 所述构造的 Voronoi 网格生成。

5.2.3. 最优控制。由偏微分方程控制的最优控制问题涉及寻找一个控制函数，在满足偏微分方程施加的约束的同时，使代价泛函最小化。我们考虑一个由泊松方程控制的分布式最优控制问题的典型示例 [28]。给定一个有界区域 $\Omega \subset \mathbb{R}^2$，其边界为 $\partial\Omega$，我们寻求使代价泛函

$$\min_{z \in Z_{ad}} J(u, z) = \frac{1}{2} \|u - u_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2} \|z\|^2_{L^2(\Omega)},$$

受状态方程约束

$$-\Delta u = z + f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega, \quad z \in Z_{ad} := \{z \in L^2(\Omega) : z_a \leqslant z(x) \leqslant z_b \text{ a.e. in } \Omega\},$$

其中 u 为状态变量，z 为控制变量，u_d 为期望状态，$\alpha > 0$ 为正则化参数，$z_a < z_b$ 为实数。众所周知，上述分布式最优控制问题等价于求解如下Karush-Kuhn-Tucker（KKT）最优性系统：$(u, p, z) \in H_0^1(\Omega) \times H_0^1(\Omega) \times Z_{ad}$

$$a(u, v) = (z + f, v), \quad a(v, p) = (u - u_d, v), \quad (p + \alpha z, w - z) \geqslant 0,$$

对所有 v，$w \in H_0^1(\Omega)$ 以及 $w \in Z_{ad}$。这里，p 是满足相应伴随方程的伴随变量。我们使用混合阶 HHO 方法，并利用控制变量 $z = \min\{z_b, \max\{z_a, -(1/\alpha)p\}\}$ 的隐式刻画来近似 KKT 系统。离散 KKT 系统如下：求 $(u_h, p_h) \in U_h^{k+1} \times U_h^{k+1} \times P_h^k$ 使得

$$a_h(\underline{u}_h, \underline{v}_h) = l_h(v_{\mathcal{T}_h}) + (\min\{z_b, \max\{z_a, -(1/\alpha)p_{\mathcal{T}_h}\}\}, v_{\mathcal{T}_h}), \quad a_h(\underline{w}_h, \underline{p}_h) = (w_{\mathcal{T}_h}, u_{\mathcal{T}_h} - u_d),$$

示例：混合阶 HHO Stokes

```
using
using       .        ,        .
using       .        ,        .
using       .            ,        .
using       .

 ( ) =            ( [1],- [2])
 ( ) =  [1] +  [2]
 ( ) = - ( )( ) -  ( )( )

      =             ((0,1,0,1),(6,6))
  =              (ReferenceFE{2},       )
  =              (ReferenceFE{1},       )

      = 1
  =          .                 (
   , VectorValue{2,Float64},       +1
)
  =          .                 (
   , VectorValue{2,Float64},       ;
                =
)
  =             ( , )

  =          .                 ( , Float64,       )
  =          .                 (
   , Float64,       ;              =
)
  =          .                 ( , Float64,  0)
  =                ( )

      =                      (2,(2,3))
   =                  ([ ,   ,   ,   ,  ];      )
   =                  ([ ,   ,   ,   ,  ];      )

      =               (      )
   =                   (      ,      )
   =                            (      ,      )

        = 2*(      +1)
   =         ( ,        )
    =        (  ,        )
    =        (  ,        )

   =                  (
  [ ,  ];      =                    (2,(1,1))
)
function          (  )
  function          ( )
    return                  (
        (                 ( ,[1,2],2)),  [1,3], 5
    )
  end
end
   =          (
                    ( ,      ,  ,      )
)
   =          (
                            ( ,      ,  ,      )
)

   =          (    .(                 (  )),     )
   =                    ( ,    ,     )

function  ((     ,      ),(      ,      ))
      =  (     ) +  (     )
      =  (     ) +  (     )
  return  (     ⊙     )
end

function  ( , )
    ((  ,   )) =   (  ) -
  return  ((1/  ) * (  ( )·  ( )))
end

 ((     ,      ), ) =  ((      +      )· )
 ( , ) =  (   )
 ((  ,  )) =  (     )

function          (      ,      , , )
     =          .            ( ,      )
   ,   =                   ( ),              ( )

   ,   ,   ,   = ( [1], [3]),   [2],   [4],   [5]
   ,   ,   ,   = ( [1], [3]),   [2],   [4],   [5]
   ,   =    +  ,     +

    ,     =  ( ),  ( )
    ,     =  ( ),  ( )

       =                                          (
          ,
    (  ,   ,  (  ,  ),                    (),      (  )),
    ( ,   ,  ( , )+ ( , )+ ( , ),   ( ),      ( )),
    ( ,    ,  (  , ),                    (),      ( )),
    (  ,   ,  (  , ),                    (),      (  ))
  )
                            (      ,     )
end

      =                (      , , )
   =          .                          (
   ,     ,        (      ,      , , )
)

   =      (  )
  ,   ,    =   [1],     [3],     [2]+  [4]

  =                          ( ,      ,  ,      )
   =                    ([   ,   ])
    =  (  )

       =     (   ( ((   -  )·(   -  ))*  ))
```

Listing 11. 不可压缩斯托克斯问题的混合阶HHO离散（第 5.2.2 节）。需要使用图 3 和清单 12 中的代码。

对所有 $v_h, w_h \in U^{k+1}_{,h}{}^{k}$。可以利用刻画 $z_{\mathcal{T}_h} = \min\{z_b, \max\{z_a, (-1/\alpha)p_{\mathcal{T}_h}\}\} \in Z_{ad}$对最优控制进行后处理。以下代码片段（见代码清单 13）展示了此示例的实现。该方法的收敛结果见图 8。

6. Conclusion

我们介绍了一个全面的框架，用于在 Gridap 库中实现非协调混合多面体有限元方法。通过引入多面体网格表示、分片多项式空间、基于片的局部组装、局部算子和静态凝聚等抽象，我们使研究人员能够以最少的代码实现复杂的混合方法，同时保持计算效率。

该框架的设计利用 Julia 的多重分派与 Gridap 的惰性求值，实现了一种高度接近数学记号的高层声明式编程模型。间断伽辽金、混合间断伽辽金以及 HHO 方法的完整实现仅需约 100 行代码，极大地

向量重构与散度算子

```
function                                  (
  ,     , ,
)
    =                ( )
    =                  (
     ,VectorValue{D,Float64},      +1
  )
    =                  (
     ,VectorValue{D,Float64},0
  )

     =                    ( ,      )
     =                            ( ,      )

          = 2*(      +1)
      =         (   ,        )
      =         (   ,        )

    =                   (   )
    ( ) =     ( )
   ( , ) =                ( , ,              ( ))
     (( , ),( , ))    =  ( ( )⊙ ( ) +      +     )
     ((   ,   ),( , )) =  ( (   )⊙ ( ) +      )     +
                          ((    -   (   ,   ))·   ( ))

        =                        (2,(1,1))
    =          .                   ( )
    =                   ([ , ];      )
    =                (
                          (),       ,    ,   ,     ,     ;
                =
  )
  return
end
function                        ( ,      , ,      )
    =                   ( ,Float64,      +1)

     =                    ( ,      )
     =                            ( ,      )

          = 2*(      +1)
      =         (   ,        )
      =         (   ,        )

    =                   (   )
   ( , ) =                ( , ,              ( ))
     (( ,),( ,))     =  (    )
     ((   ,   ),( ,)) =  ((     )·  )     +
                         ((    -   (   ,   ))·   *   )

        =                        ()
    =          .                   ( )
    =                   ([ ];      )
    =                (
                  (),       ,    ,   ,     ,     ;
                =
  )
  return
end
```

Listing 12.不可压缩斯托克斯问题的混合高阶方法局部算子（第 5.2.2 节）：速度重构和散度算子。

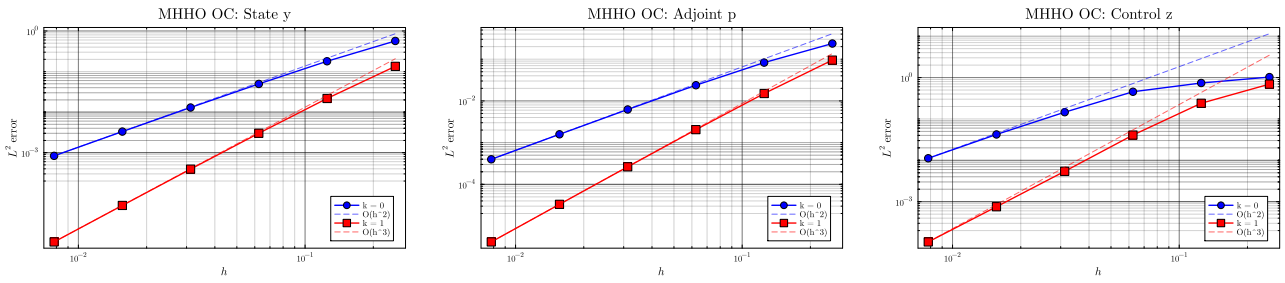

Figure 8. 最优控制问题的混合阶 HHO 方法收敛性：L^2-误差与网格尺寸 h 的关系，多项式次数为 $k = 0$，1。状态变量 u（左）、伴随变量 p（中）和控制变量 z（右）。参考线表示预期收敛速率 $O(h^{k+2})$。结果由代码清单 13 中的驱动程序生成，使用按图 3 所述构建的沃罗诺伊网格。

降低了相较于传统方法的实现负担。尽管如此简短，所生成的代码仍能借助即时编译达到与手工优化实现相当的性能。尽管本文并未讨论，该框架还通过 GridapDistributed 支持分布式内存并行 [8]并通过 GridapSolvers 支持可扩展求解器 [25]，从而能够在多面体网格上进行大规模模拟 [5, 7]。

第 5 节中的数值算例表明，该框架能够成功处理复杂程度各异的方法，从简单的 DG 离散化到涉及多个局部重构算子的复杂 HHO 公式。收敛结果证实，这些实现能够在多面体网格上达到最优速率，从而验证了数学公式和计算框架。

除了本文展示的具体方法外，本研究引入的抽象概念具有足够的通用性，可支持更广泛的一类多面体离散化。斑块组装框架

示例：混合阶 HHO 最优控制

```
using
using       .        ,        .
using       .        ,        .
using       .            ,        .
using       .      ,        .

   ( ) =    (2 * [1])*   (2 * [2])
   ( ) = -10*   (2*  * [1])*   (2*  * [2])*
             [1]*( [1]-1)* [2]*( [2]-1)
   = -1;    = 1;   = 0.01
   ( ) =    (   ,    (  , -(1/ ) *    ( )) )
 ( ) = - (   )( ) -    ( )
  ( ) =    ( ) +  (   )( )

      =            ((0,1,0,1), (8,8))
  =              (ReferenceFE{2},      )
  =              (ReferenceFE{1},      )

      =              (     )
   =                  (     ,     )
   =                          (     ,     )

      = 1
        = 2*(     +1)
    =        (  ,       )
    =        (  ,       )
# State
   =                ( , Float64,       +1)
   =                ( , Float64,       ;
                              =          )
   =                ( , Float64,       +1)
   =             (  ,    )
# Adjoint
   =                ( , Float64,       +1)
   =                ( , Float64,       ;
                              =          )
   =                ( , Float64,       +1)
       =            (  ,    )

      =           .                    (2,(1,1))
   =                 ([  ,   ];     =     )
   =                 ([  ,   ];     =     )
   =                 ([  ,   ];     =     )
   =                 ([  ,   ];     =     )

    =                   (  ,   ,    )
    =                   (  ,   ,    )
    =                       ( ,      ,   ,      )
    =                       ( ,      ,   ,      )

function     ( , ) return ( + +   ( - ))/2 end

function     ( , ) return ( + -   ( - ))/2 end

     ( ) =     (   ,     (   , (-1/ )*  ) )

function  ((   ,   ),(   ,   ))
      =  (   ) +  (   )
      =  (   ) +  (   )
  return  (    ·    )
end

# Mixed-order stabilisation
   =          (    .(                (  )),   )
function   ( , )
  function    ( )
       ,     =
    return    (   ) -
  end
  return  ((1/  ) * (   ( )·   ( )))
end

function   ( , )
  function    ( )
       ,     =
    return    (   ) -
  end
  return  ((1/  ) * (   ( )·   ( )))
end

  (( ,  ),  ) =  ( *  )    +  (     (  )*  )
  (( ,  ),  ) =  ( (  -  )*   )

function                 ( , ,  )
   ,   =                  ( ),              ( )
     =              ( ,     )
    ,    =   ( ),   ( )
     =                  ()
     =              ( ,     )
       =                                        (
          , ( ,  ,   ( , ),    ( , ),      ( )),
    (  ,   ,  (  ,  ),    ,     ( ))
  )
  return                          (       ,    )
end

function                 ( , ,  )
   ,    =                  ( ),              ( )
     =              ( ,     )
    ,    =   ( ) ,   ( )
     =                  ()
     =              ( ,     )
       =                                        (
          , ( ,  ,   ( , ),    ( , ),      ( )),
    (  ,   ,  (  ,  ),    ,     ( ))
  )
  return                          (       ,    )
end

        =              (     ,  ,  )
        =              (     ,  ,  )

# Initialize and solve KKT
   =           (  [1],      (             (  [1])))
    = 1e-6,          = 25
function          (  ,   ,        )
       = 0
     = 1.0
     =        ,    =
     =        ,    =
  while (   >    ) && (     <          )
         += 1
         =                          (
      ,  ,  ,       ,                (  ,  ,  ))
       =      (    .     )
       =                            (    ,   )
         =                          (
      ,  ,  ,       ,                (  ,  ,  ))
       =      (    .     )
       =                            (    ,   )
       =    (    (  ( (   -   )·(   -   ) )    ) )
          (                      )
       =
  end

  return   ,   ,   ,

end
# optimal state, adjoint and control
    =          (  ,   ,        )
  ,  ,   ,   =    ;
  =      (  )

  =    -
  =    -
  =     -
     =     (    (  (   *   )*    ) )
     =     (    (  (   *   )*    ) )
     =     (    (  (   *   )*    ) )
```

Listing 13. 最优控制问题的混合阶HHO离散（第 5.2.3 节）。需要图 3 和代码清单 4 中的代码。

能够容纳重叠块和多单元配置，从而支持基于补丁的多重网格光滑器及其他先进技术的应用。

未来的工作将侧重于扩展该框架，以支持更多混合方法变体，例如虚拟元方法或多面体网格上的离散 de Rham 序列 [18]。

Acknowledgments

本研究部分由澳大利亚政府通过澳大利亚研究理事会资助（项目编号 DP210103092 和 DP220103160）。

References

[1] J. Aghili、S. Boyaval 和 D. A. D. Pietro。“一般网格上的混合高阶方法杂交化及其在斯托克斯方程中的应用”。载于：《应用数学计算方法》15.2（2015），第 111–134 页。 DOI: doi:10.1515/cmam-2015-0004。[2] R. Anderson 等。“MFEM：模块化有限元方法库”。载于：Computers & Mathematics with Applications 81（2021），第 42–74 页。 DOI: 10.1016/j.camwa.2020.06.009。[3] P. F. Antonietti 等。“lymph：用于多物理场微分问题的间断多面体方法”。载于：《ACM 数学软件汇刊》51.1（2025），第 1–22 页。[4] D. Arndt 等。“deal.II 库，版本 9.3”。载于：《数值数学杂志》29.3（2021），第 171–186 页。 DOI: 10.1515/jnma-2021-0081。[5] S. Badia、J. Droniou、J. Manyer 和 J. Tushar。非协调多面体杂交离散方法的 BDDC 预条件器分析。2025。arXiv：2506。11956 [math.NA]。[6] S. Badia、J. Droniou 和 L. Yemm。“具有小面的网格上混合高阶格式的条件数”。载于：《科学计算杂志》92.2（2022年7月），第 71 页。 DOI：10.1007/s10915-022-01913-9。[7] S. Badia 和 J. Manyer。“多面体网格上混合高阶方法的几何多重网格求解器”。准备中的手稿。2026年。[8] S. Badia、A. F. Mart´ın 和 F. Verdugo。“GridapDistributed：Julia 中的大规模并行有限元工具箱”。载于：《开放源代码软件杂志》7.74（2022），第 4157 页。 DOI：10.21105/joss.04157。[9] S. Badia、P. A. Martorell 和 F. Verdugo。“显式边界表示上非拟合有限元的几何离散”。载于：《计算物理杂志》460（2022），第 111162 页。 DOI：https://doi.org/10.1016/j.jcp.2022.111162。[10] S. Badia 和 F. Verdugo。“Gridap：Julia 中可扩展的有限元工具箱”。载于：《开放源代码软件杂志》5.52（2020），第 2520 页。 DOI：10.21105/joss.02520。[11] S. Berrone、A. Borio、G. Teora 和 F. Vicini。“POLYDIM：用于多面体离散方法的 C++库”。载于：《计算物理通讯》320（2025），第 109937 页。 DOI：10.1016/j.cpc.2025.109937。[12] J. Bezanson、A. Edelman、S. Karpinski 和 V. B. Shah。“Julia：数值计算的新方法”。载于：《SIAM 评论》59.1（2017），第 65–98 页。 DOI: 10.1137/141000671。[13] M. Blatt 等。“分布式与统一数值环境（DUNE），版本 2.10”。载于：arXiv 预印本 arXiv:2506.23558（2025）。[14] A. Cangiani、E. H. Georgoulis 和 P. Houston。“多边形和多面体网格上的 hp 版本间断伽辽金方法”。载于：Mathematical Models and Methods in Applied Sciences 24.10（2014），第 2009–2041 页。[15] B. Cockburn、J. Gopalakrishnan 和 R. Lazarov。“用于二阶椭圆问题的间断伽辽金、混合和连续伽辽金方法的统一杂交化”。载于：《SIAM 数值分析杂志》（2009）。 DOI：10.1137/070706616。[16] L. B. Da Veiga、F. Brezzi、L. D. Marini 和 A. Russo。“虚拟元方法”。载于：《数值纪事》32（2023），第 123–202 页。[17] F. Dassi。“Vem++，用于处理和探索虚拟元方法的 C++ 库”。载于：《数值算法》（2025），第 1–43 页。[18] D. A. Di Pietro、J. Droniou 和 F. Rapetti。“多边形和多面体上任意次数的完全离散多项式 de Rham 序列”。载于：Mathematical Models and Methods in Applied Sciences 30.09（2020年8月），第 1809–1855 页。 DOI：10.1142/s0218202520500372。

[19] D. A. Di Pietro 和 A. Ern。“一般网格上线弹性的混合高阶无锁方法”。载于：Computer Methods in Applied Mechanics and Engineering 283（2015），第 1–21 页。 DOI: https://doi.org/10.1016/j.cma.2014.09.009。[20] S. Du 和 F.-J. Sayas。“关于在多面体单元上构造混合间断伽辽金投影的说明”。载于：arXiv：数值分析（2020）。+ ： 10.1090/mcom/3573。 DOI P. E. Farrell、M. G. Knepley、L. Mitchell 和 F. Wechsung。“PCPATCH：用于多重网格松弛方法拓扑构造的软件”。载于：《ACM 数学软件汇刊》47.3（2021年6月）。[21] ： 10.1145/3445791。 DOIM. Feder、A. Cangiani 和 L. Heltai。“R3MG：基于 R 树的多面体网格聚合及其在多层方法中的应用”。载于：《计算物理杂志》526（2025），第 113773 页。[22] ： https://doi.org/10.1016/j.jcp.2025.113773。 DOIGridap有限元库教程。https:/ /gridap. github.io / Tutorials/ stable/。[23] A. Logg、K.-A. Mardal 和 G. Wells 编。《有限元方法求解微分方程》。Springer Berlin Heidelberg，2012。[24] ： 10.1007/978-3-642-23099-8。 DOIJ. Manyer、A. F. Mart´ın 和 S. Badia。“GridapSolvers.jl：Julia 中可扩展的多物理场有限元求解器”。载于：《开放源代码软件杂志》9.102（2024），第 7162 页。 DOI: 10.21105/joss.07162。[26] D. A. D. Pietro 和 J. Droniou。《多面体网格的混合高阶方法：设计、分析与应用》。第 19 卷。MS&A。Springer，2020。 DOI: 10.1007/978-3-030-37203-3。[27] J. Sch¨oberl。《NGSolve 中有限元的 C*++11* 实现》。技术报告 30/2014。维也纳工业大学分析与科学计算研究所，2014。[28] F. Tr¨oltzsch。《偏微分方程最优控制：理论、方法与应用》。第 112 卷。数学研究生教材。罗得岛州普罗维登斯：美国数学学会，2010。[29] F. Verdugo 和 S. Badia。“Gridap 的软件设计：基于 Julia 即时编译器的有限元软件包”。载于：《计算物理通讯》276（2022年7月），第 108341 页。 DOI: 10.1016/j.cpc.2022.108341。[30] J. Wang 和 X. Ye。“用于二阶椭圆问题的弱Galerkin有限元方法”。载于：Journal of Computational and Applied Mathematics 241（2013），第 103–115 页。