Lowest order methods for diffusive problems on general meshes: A unified approach to definition and implementation

Daniele A. Di Pietro and Jean-Marc Gratien

Abstract In this work we propose an original point of view on lowest order methods for diffusive problems which lays the pillars of a `C++` multi-physics, `FreeFEM`-like platform. The key idea is to regard lowest order methods as (Petrov)-Galerkin methods based on possibly incomplete, broken polynomial spaces defined from a gradient reconstruction. After presenting some examples of methods entering the framework, we show how implementation strategies common in the finite element context can be extended relying on the above definition. Several examples are provided throughout the presentation, and programming details are often omitted to help the reader unfamiliar with advanced `C++` programming techniques.

Keywords Lowest-order methods, Domain specific embedded language, Petrov-Galerkin methods, cell centered Galerkin methods, hybrid finite volume methods
MSC2010: 65Y99, 65N08, 65N30

1 Introduction

An increasing amount of attention has recently been given to the discretization of diffusive problems on general meshes. Lowest order methods possibly featuring conservation of physical quantities are traditionally employed in industrial applications where computational cost is a crucial issue. In this context, the main interest of handling general meshes is to reduce the number of elements required to represent complicate domains. In sedimentary basin modeling, non-standard elements may also appear due to the erosion of geological layers. Different ways to adapt finite volume and finite element methods to general, possibly non-conforming

Daniele A. Di Pietro and Jean-Marc Gratien
IFP Energies nouvelles, e-mail: dipietrd@ifpenergiesnouvelles.fr, j-marc.gratien@ifpenergiesnouvelles.fr

polyhedral meshes have been proposed. In the context of cell centered finite volume methods, we recall, in particular, the classical works of Aavatsmark, Barkve, Bøe and Mannseth [1] and Edwards and Rogers [15] on multipoint fluxes. More recently, two ways of extending the mixed finite element philosophy to general meshes have been proposed independently by Brezzi, Lipnikov, Shashkov and Simoncini [5, 6] (mimetic finite difference methods) and by Droniou and Eymard [13] (mixed/hybrid finite volume methods). Yet another perspective is considered by Eymard, Gallouët and Herbin [17], who show, in particular, that face unknowns can be selectively used as Lagrange multipliers for the flux continuity constraint or be eliminated using a consistent interpolator (SUSHI scheme). The strong link between the strategies above has been highlighted by Droniou, Eymard, Gallouët and Herbin [14]. A slightly different approach based on the analogy between lowest order methods in variational formulation and discontinuous Galerkin methods has been proposed by the author in [8–10] (cell centered Galerkin methods). The key advantage of this approach is that it largely benefits from the well-established theory for discontinuous Galerkin methods applied to diffusive problems [11]. All of the methods above have been (or can be) extended to several classical problems for which the discretization of second order diffusive terms is central.

In this work we present a unified implementation covering a wide range of lowest order methods and applications based on similar experiences in the context of finite element methods. Finite element libraries have nowadays reached a good level of maturity, and user-friendly front-ends are provided in several cases. Just to mention a few, we recall `Feel++` [20] (formerly known as `Life`), `FEniCS` [19], `FreeFEM++` [7]. Our goal is to show that similar tools can be conceived and implemented for lowest order methods. The starting point is to reformulate the method at hand as a (Petrov)-Galerkin scheme based on possibly incomplete broken affine spaces. This new unified perspective, drawing on the lines of [9], allows, in particular, to recycle many ideas originally developed for finite elements. A major difference, however, is that the lowest order methods considered herein are often based on reconstructions of first order differential operators which may depend on problem data such as the diffusion coefficient or the boundary condition. As a consequence, the classical approach based on a table of degrees of freedom computed from a mesh and a finite element (see, e.g., [16, Chapters 7–8]) is no longer adequate. This issue is solved by introducing the programming counterpart of tensor-valued linear combinations of (globally numbered) degrees of freedom. This concept allows, in particular, to reproduce a finite element-like matrix assembly with local contributions stemming from integrals over mesh elements and faces. A further layer of abstraction is added by defining a domain-specific language (DSL) for variational formulations. The DSL is closely inspired by that of `Feel++`, the most noticeable differences being the type-based identification of test and trial functions and the possibility to store the expressions defining linear and bilinear forms independently of their algebraic representation. Another novelty is the introduction of tensor-like notation for systems of PDEs. Domain-specific languages and generative programming are an established tool to break down the complexity of industrial applications by distinguishing the actors that tackle different aspects of the problem, and providing each of them with means of expression as close as possible to his/her technical jargon. An important advantage of the DSL is that it potentially allows to combine lowest order methods with more standard discretizations techniques in a seamless way. In the presentation we try to avoid all technicalities and to pinpoint the main difficulties as well as the proposed solutions. Although the language of choice is `C++`, the listings are rather to be intended as pseudo-code since simplifications are often made to improve readability. The actual implementation is based on the `Arcane` framework [18], a proprietary platform conjointly developed at CEA-DAM and IFP Energies nouvelles which takes care of technical aspects such as memory management, parallelism and post-processing.

The material is organized as follows. In §2 we propose a unified perspective and show how several lowest order methods can fit in there for a simple diffusion problem. In §3 we discuss the implementation. More specifically, we first discuss the solutions to the issues that arise when trying to mimic the finite element approach and then present a DSL which allows to conceal the related technicalities.

2 Definition

2.1 Discrete setting

Let $\Omega \subset \mathbb{R}^d$, $d \geq 2$, denote a bounded connected polyhedral domain. The first ingredient in the definition of lowest order methods is a suitable discretization of Ω. We denote by $\mathcal{T}_h$ a finite collection of nonempty, disjoint open polyhedra $\mathcal{T}_h = \{T\}$ forming a partition of Ω such that $h = \max_{T\in\mathcal{T}_h} h_T$ and h_T denotes the diameter of the element $T \in \mathcal{T}_h$. Admissible meshes include general polyhedral discretizations with possibly nonconforming interfaces; see Fig. 1. Mesh nodes are collected in the set $\mathcal{N}_h$ and, for all $T \in \mathcal{T}_h$, $\mathcal{N}_T$ contains the nodes that lie on the boundary of T. We say that a hyperplanar closed subset F of $\overline{\Omega}$ is a mesh face if it has positive $(d-1)$-dimensional measure and if either there exist $T_1, T_2 \in \mathcal{T}_h$ such that $F \subset \partial T_1 \cap \partial T_2$ (and F is called an *interface*) or there exists $T \in \mathcal{T}_h$ such that $F \subset \partial T \cap \partial\Omega$ (and F is called a *boundary face*). Interfaces are collected in the set $\mathcal{F}_h^{\mathrm{i}}$, boundary faces in $\mathcal{F}_h^{\mathrm{b}}$ and we let $\mathcal{F}_h := \mathcal{F}_h^{\mathrm{i}} \cup \mathcal{F}_h^{\mathrm{b}}$. For all $T \in \mathcal{T}_h$ we set

$$\mathcal{F}_T := \{F \in \mathcal{F}_h \mid F \subset \partial T\}. \tag{1}$$

Symmetrically, for all $F \in \mathcal{F}_h$, we define

$$\mathcal{T}_F := \{T \in \mathcal{T}_h \mid F \subset \partial T\}.$$

The set $\mathcal{T}_F$ consists of exactly two mesh elements if $F \in \mathcal{F}_h^{\mathrm{i}}$ and of one if $F \in \mathcal{F}_h^{\mathrm{b}}$. For all mesh nodes $P \in \mathcal{N}_h$, $\mathcal{F}_P$ denotes the set of mesh faces sharing P, i.e.

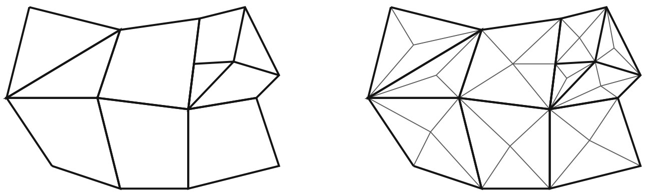

Fig. 1 *Left.* Mesh $\mathscr{T}_h$ *Right.* Pyramidal submesh $\mathscr{P}_h$

$$\mathscr{F}_P := \{F \in \mathscr{F}_h \mid P \in F\}. \tag{2}$$

The diameter of a face $F \in \mathscr{F}_h$ is denoted by h_F. For every interface $F \in \mathscr{F}_h^{\mathrm{i}}$ we introduce an arbitrary but fixed ordering of the elements in $\mathscr{T}_F$ and let $\mathbf{n}_F = \mathbf{n}_{T_1,F} = -\mathbf{n}_{T_2,F}$, where $\mathbf{n}_{T_i,F}$, $i \in \{1,2\}$, denotes the unit normal to F pointing out of $T_i \in \mathscr{T}_F$. On a boundary face $F \in \mathscr{F}_h^{\mathrm{b}}$, $\mathbf{n}_F$ denotes the unit normal pointing out of Ω. The barycenter of a face $F \in \mathscr{F}_h$ is denoted by $\overline{\mathbf{x}}_F := \int_F \mathbf{x}/|F|_{d-1}$. For each $T \in \mathscr{T}_h$ we identify a point $\mathbf{x}_T \in T$ (the *cell center*) such that T is star-shaped with respect to $\mathbf{x}_T$. For all $F \in \mathscr{F}_T$ we let

$$d_{T,F} := dist(\mathbf{x}_T, F).$$

It is assumed that, for all $T \in \mathscr{T}_h$ and all $F \in \mathscr{F}_T$, $d_{T,F} > 0$ is comparable to h_T. Starting from cell centers we can define a pyramidal submesh of $\mathscr{T}_h$ as follows:

$$\mathscr{P}_h := \{\mathscr{P}_{T,F}\}_{T\in\mathscr{T}_h,\, F\in\mathscr{F}_T},$$

where, for all $T \in \mathscr{T}_h$ and all $F \in \mathscr{F}_T$, $\mathscr{P}_{T,F}$ denotes the open pyramid of apex $\mathbf{x}_T$ and base F, i.e.,

$$\mathscr{P}_{T,F} := \{\mathbf{x} \in T \mid \exists \mathbf{y} \in F \setminus \partial F,\ \exists \theta \in (0,1) \mid \mathbf{x} = \theta\mathbf{y} + (1-\theta)\mathbf{x}_T\}.$$

The pyramids $\{\mathscr{P}_{T,F}\}_{T\in\mathscr{T}_h,\, F\in\mathscr{F}_T}$ are nondegenerate by assumption. Let $\mathscr{S}_h$ be such that

$$\mathscr{S}_h = \mathscr{T}_h \text{ or } \mathscr{S}_h = \mathscr{P}_h. \tag{3}$$

For all $k \geq 0$, we define the broken polynomial spaces of total degree $\leq k$ on $\mathscr{S}_h$,

$$\mathbb{P}_d^k(\mathscr{S}_h) := \{v \in L^2(\Omega) \mid \forall S \in \mathscr{S}_h,\ v_{|S} \in \mathbb{P}_d^k(S)\},$$

with $\mathbb{P}_d^k(S)$ given by the restriction to $S \in \mathscr{S}_h$ of the functions in $\mathbb{P}_d^k$.

Remark 1 (Admissible mesh sequence). In the context of *a priori* convergence analysis for vanishing mesh size h it is necessary to bound some quantities uniformly with respect to h. This leads to the concept of *admissible mesh sequence*. This topic is not addressed in detail herein since our focus is mainly on implementation. For a comprehensive discussion we refer to [5,6,9,13,17]; see also [11, Chapter 1].

We close this section by introducing trace operators which are of common use in the context of nonconforming finite element methods. Let v be a scalar-valued function defined on Ω smooth enough to admit on all $F \in \mathscr{F}_h$ a possibly two-valued trace. To any interface $F \subset \partial T_1 \cap \partial T_2$ we assign two nonnegtive real numbers $\omega_{T_1,F}$ and $\omega_{T_2,F}$ such that

$$\omega_{T_1,F} + \omega_{T_2,F} = 1,$$

and define the jump and weighted average of v at F for a.e. $\mathbf{x} \in F$ as

$$[\![v]\!]_F(\mathbf{x}) := v_{|T_1} - v_{|T_2}, \qquad \{\!\{v\}\!\}_{\omega,F}(\mathbf{x}) := \omega_{T_1,F} v_{|T_1}(\mathbf{x}) + \omega_{T_2,F} v_{|T_2}(\mathbf{x}). \tag{4}$$

If $F \in \mathscr{F}_h^{\mathrm{b}}$ with $F = \partial T \cap \partial\Omega$, we conventionally set $\{\!\{v\}\!\}_{\omega,F}(\mathbf{x}) = [\![v]\!]_F(\mathbf{x}) = v_{|T}(\mathbf{x})$. The subscript ω is omitted from the average operator when $\omega_{T_1,F} = \omega_{T_2,F} = \frac{1}{2}$. The dependence on $\mathbf{x}$ and on the face F is also omitted if no ambiguity arises.

2.2 An abstract perspective

The key idea to gain a unifying perspective is to regard lowest order methods as nonconforming methods based on incomplete broken affine spaces that are defined starting from the space of degrees of freedom (DOFs) $\mathbb{V}_h$. More precisely, we let

$$\mathbb{T}_h := \mathbb{R}^{\mathscr{T}_h}, \qquad \mathbb{F}_h := \mathbb{R}^{\mathscr{F}_h},$$

and consider the following choices:

$$\mathbb{V}_h = \mathbb{T}_h \text{ or } \mathbb{V}_h = \mathbb{T}_h \times \mathbb{F}_h.$$

In every case the elements of $\mathbb{V}_h$ are indexed with respect to the mesh entity they belong to. Other choices for $\mathbb{V}_h$ are possible but are not considered herein for the sake of conciseness. To fix the ideas, one can assume that the choice $\mathbb{V}_h = \mathbb{T}_h$ corresponds to cell centered finite volume (CCFV) and cell centered Galerkin (CCG) methods, while the choice $\mathbb{V}_h = \mathbb{T}_h \times \mathbb{F}_h$ leads to mimetic finite difference (MFD) and mixed/hybrid finite volume (MHFV) methods.

The key ingredient in the definition of the broken affine space is a piecewise constant linear gradient reconstruction $\mathfrak{G}_h : \mathbb{V}_h \to [\mathbb{P}_d^0(\mathscr{S}_h)]^d$ (the linearity of $\mathfrak{G}_h$ is a founding assumption for the implementation discussed in §3). Starting from $\mathfrak{G}_h$, we can define the linear operator $\mathfrak{R}_h : \mathbb{V}_h \to \mathbb{P}_d^1(\mathscr{S}_h)$ such that, for all $\mathbf{v}_h \in \mathbb{V}_h$,

$$\forall S \in \mathscr{S}_h,\ S \subset T_S \in \mathscr{T}_h,\ \forall \mathbf{x} \in S, \quad \mathfrak{R}_h(\mathbf{v}_h)_{|S} = v_{T_S} + \mathfrak{G}_h(\mathbf{v}_h)_{|S} \cdot (\mathbf{x} - \mathbf{x}_{T_S}) \in \mathbb{P}_d^1(\mathscr{S}_h). \tag{5}$$

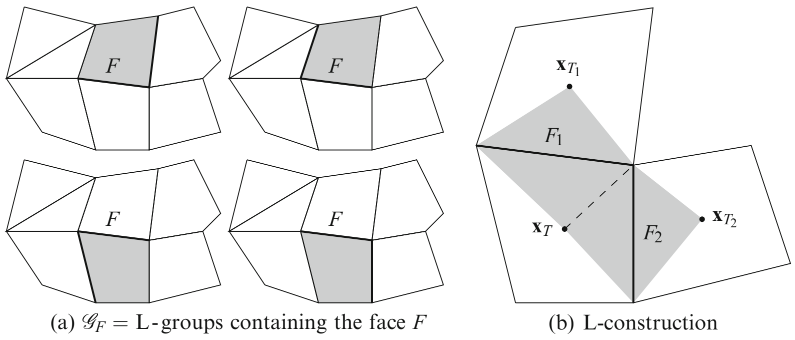

Fig. 2 L-construction

The operator $\mathfrak{R}_h$ maps every vector of DOFs onto a piecewise affine function belonging to $\mathbb{P}^1_d(\mathscr{S}_h)$. Hence, we can define a broken affine space as follows:

$$V_h = \mathfrak{R}_h(\mathbb{V}_h) \subset \mathbb{P}^1_d(\mathscr{S}_h). \tag{6}$$

The operator $\mathfrak{R}_h$ is additionally assumed to be injective, so that a bijective operator can be obtained by restricting its codomain. The next section presents some examples covering the methods listed above.

2.3 Examples

In this section we focus on the model problem

$$-\nabla\cdot(\boldsymbol{\kappa}\nabla u) = f, \qquad u = 0, \tag{7}$$

where $f \in L^2(\Omega)$ and $\boldsymbol{\kappa} \in [\mathbb{P}^0_d(\mathscr{T}_h)]^d$ is a piecewise constant, uniformly elliptic tensor field (possibly resulting from a homogeneization process). Problem (7) provides the paradigm to illustrate how selected lowest order methods can be recast in the framework of §2.2.

The G-method As a first example we consider the special instance of CCFV methods analyzed in [3]. A preliminary step consists in presenting the so-called L-construction introduced in [2]. The key idea of the L-construction is to use d cell and boundary face values (provided, in this case, by the homogeneous boundary condition) to express a continuous piecewise affine function with continuous diffusive fluxes. The values are selected using d neighboring faces belonging to a cell and sharing a common vertex. More precisely, we define the set of L-groups (see Fig. 2) as follows:

$$\mathscr{G} := \{\mathfrak{g} \subset \mathscr{F}_T \cap \mathscr{F}_P,\ T \in \mathscr{T}_h,\ P \in \mathscr{N}_T \mid card(\mathfrak{g}) = d\},$$

with $\mathscr{F}_T$ and $\mathscr{F}_P$ given by (1) and (2) respectively. It is useful to introduce a symbol for the set of cells concurring in the L-construction: For all $\mathfrak{g} \in \mathscr{G}$, we let

$$\mathscr{T}_{\mathfrak{g}} := \{T \in \mathscr{T}_h \mid T \in \mathscr{T}_F,\ F \in \mathfrak{g}\}.$$

Let now $\mathfrak{g} \in \mathscr{G}$ and denote by $T_{\mathfrak{g}}$ an element $T_{\mathfrak{g}}$ such that $\mathfrak{g} \subset \mathscr{F}_{T_{\mathfrak{g}}}$ (this element may not be unique). For all $\mathbf{v}_h \in \mathbb{V}_h$ we construct the function $\xi^{\mathfrak{g}}_{\mathbf{v}_h}$ piecewise affine on the family of pyramids $\{\mathscr{P}_{T,F}\}_{F\in\mathfrak{g},\,T\in\mathscr{T}_{\mathfrak{g}}}$ such that: (i) $\xi^{\mathfrak{g}}_{\mathbf{v}_h}(\mathbf{x}_T) = v_T$ for all $T \in \mathscr{T}_{\mathfrak{g}}$ and $\xi^{\mathfrak{g}}_{\mathbf{v}_h}(\overline{\mathbf{x}}_F) = 0$ for all $F \in \mathfrak{g} \cap \mathscr{F}^{\mathrm{b}}_h$; (ii) $\xi^{\mathfrak{g}}_{\mathbf{v}_h}$ is affine inside $T_{\mathfrak{g}}$ and is continuous across every interface in the group: For all $F \in \mathfrak{g} \cap \mathscr{F}^{\mathrm{i}}_h$ such that $F \subset \partial T_1 \cap \partial T_2$,

$$\forall \mathbf{x} \in F, \qquad \xi^{\mathfrak{g}}_{\mathbf{v}_h|T_1}(\mathbf{x}) = \xi^{\mathfrak{g}}_{\mathbf{v}_h|T_2}(\mathbf{x});$$

(iii) $\xi^{\mathfrak{g}}_{\mathbf{v}_h}$ has continuous diffusive flux across every interface in the group: For all $F \in \mathfrak{g} \cap \mathscr{F}^{\mathrm{i}}_h$ such that $F \subset \partial T_1 \cap \partial T_2$,

$$(\boldsymbol{\kappa}\nabla\xi^{\mathfrak{g}}_{\mathbf{v}_h})_{|T_1}\cdot\mathbf{n}_F = (\boldsymbol{\kappa}\nabla\xi^{\mathfrak{g}}_{\mathbf{v}_h})_{|T_2}\cdot\mathbf{n}_F.$$

For further details on the L-construction including an explicit formula for $\xi^{\mathfrak{g}}_{\mathbf{v}_h}$ we refer to [3]. For every face $F \in \mathscr{F}_h$ we define the set $\mathscr{G}_F$ of L-groups containing F,

$$\mathscr{G}_F := \{\mathfrak{g} \in \mathscr{G} \mid F \in \mathfrak{g}\}, \tag{8}$$

and introduce the set of nonnegative weights $\{\varsigma_{\mathfrak{g},F}\}_{\mathfrak{g}\in\mathscr{G}_F}$ such that $\sum_{\mathfrak{g}\in\mathscr{G}_F} \varsigma_{\mathfrak{g},F} = 1$. The trial space for the G-method is obtained as follows: (i) let $\mathscr{S}_h = \mathscr{P}_h$ and $\mathbb{V}_h = \mathbb{T}_h$; (ii) let $\mathfrak{G}_h = \mathfrak{G}^{\mathrm{g}}_h$ with $\mathfrak{G}^{\mathrm{g}}_h$ such that

$$\forall \mathbf{v}_h \in \mathbb{T}_h,\ \forall T \in \mathscr{T}_h,\ \forall F \in \mathscr{F}_T, \qquad \mathfrak{G}^{\mathrm{g}}_h(\mathbf{v}_h)_{|\mathscr{P}_{T,F}} = \sum_{\mathfrak{g}\in\mathscr{G}_F} \varsigma_{\mathfrak{g},F}\nabla\xi^{\mathfrak{g}}_{\mathbf{v}_h|\mathscr{P}_{T,F}}.$$

We denote by $\mathfrak{R}^{\mathrm{g}}_h$ the reconstruction operator defined as in (5) with $\mathfrak{G}_h = \mathfrak{G}^{\mathrm{g}}_h$ and let $V^{\mathrm{g}}_h := \mathfrak{R}^{\mathrm{g}}_h(\mathbb{V}_h)$. The G-method of [3] is then equivalent to the following Petrov-Galerkin method:

$$\text{Find } u_h \in V^{\mathrm{g}}_h \text{ s.t. } a^{\mathrm{g}}_h(u_h, v_h) = \int_\Omega f v_h \text{ for all } v_h \in \mathbb{P}^0_d(\mathscr{T}_h),$$

where $a^{\mathrm{g}}_h(u_h, v_h) := -\sum_{F\in\mathscr{F}_h}\int_F \{\!\{\boldsymbol{\kappa}\nabla_h u_h\}\!\}\cdot\mathbf{n}_F[\![v_h]\!]$ with ∇_h broken gradient on $\mathscr{S}_h$.

Remark 2 (An unconditionally stable method). The main drawback of the G-method is that stability can only be proven under quite stringent conditions; see, e.g., [3, Lemma 3.4]. A possible way to circumvent this difficulty has recently been proposed by one of the authors [10] in the context of CCG methods. The key idea is to use V^{g}_h both as a trial and test space, and modify the discrete bilinear form to recover both consistency and stability. Since the discrete functions in V_h^{g} are discontinuous across the lateral faces of the pyramids in $\mathscr{P}_h$, least-square penalization of the jumps is required to assert stability in terms of coercivity. The resulting method also enters the present framework, but is not detailed here for the sake of conciseness.

A cell centered Galerkin method The L-construction is used to define a trace reconstruction in the CCG method of [8, 10]. More specifically, for all $F \in \mathscr{F}_h^{\mathrm{i}}$, we select one group $\mathfrak{g}_F \in \mathscr{G}_F$ with $\mathscr{G}_F$ defined by (8) and introduce the linear trace operator $\mathbf{T}_h^{\mathrm{g}} : \mathbb{T}_h \to \mathbb{F}_h$ which maps every vector of cell centered DOFs $\mathbf{v}_h \in \mathbb{T}_h$ onto a vector $(v_F)_{F\in\mathscr{F}_h} \in \mathbb{F}_h$ such that

$$v_F = \begin{cases} \xi_{\mathbf{v}_h}^{\mathfrak{g}_F}(\overline{\mathbf{x}}_F) & \text{if } F \in \mathscr{F}_h^{\mathrm{i}}, \\ 0 & \text{if } F \in \mathscr{F}_h^{\mathrm{b}}. \end{cases} \tag{9}$$

The trace operator $\mathbf{T}_h^{\mathrm{g}}$ is then employed in a gradient reconstruction based on Green's formula and inspired from [17]. More precisely, we introduce the linear gradient operator $\mathfrak{G}_h^{\mathrm{green}} : \mathbb{T}_h \times \mathbb{F}_h \to [\mathbb{P}_d^0(\mathscr{T}_h)]^d$ such that, for all $(\mathbf{v}^{\mathscr{T}}, \mathbf{v}^{\mathscr{F}}) \in \mathbb{T}_h \times \mathbb{F}_h$ and all $T \in \mathscr{T}_h$,

$$\mathfrak{G}_h^{\mathrm{green}}(\mathbf{v}^{\mathscr{T}}, \mathbf{v}^{\mathscr{F}})_{|T} = \frac{1}{|T|_d} \sum_{F\in\mathscr{F}_T} |F|_{d-1}(v_F - v_T)\mathbf{n}_{T,F}. \tag{10}$$

The discrete space for the CCG method under examination can then be obtained as follows: (i) let $\mathscr{S}_h = \mathscr{T}_h$ and $\mathbb{V}_h = \mathbb{T}_h$; (ii) let $\mathfrak{G}_h = \mathfrak{G}_h^{\mathrm{ccg}}$ with $\mathfrak{G}_h^{\mathrm{ccg}}$ such that

$$\forall \mathbf{v}_h \in \mathbb{V}_h, \qquad \mathfrak{G}_h^{\mathrm{ccg}}(\mathbf{v}_h) = \mathfrak{G}_h^{\mathrm{green}}(\mathbf{v}_h, \mathbf{T}_h^{\mathrm{g}}(\mathbf{v}_h)). \tag{11}$$

The reconstruction operator defined taking $\mathfrak{G}_h = \mathfrak{G}_h^{\mathrm{ccg}}$ in (5) is denoted by $\mathfrak{R}_h^{\mathrm{ccg}}$, and the corresponding discrete space by $V_h^{\mathrm{ccg}} := \mathfrak{R}_h^{\mathrm{ccg}}(\mathbb{T}_h)$. We define the weights in the average operator as follows: For all $F \in \mathscr{F}_h^{\mathrm{i}}$ such that $F \subset \partial T_1 \cap \partial T_2$,

$$\omega_{T_1,F} = \tfrac{\lambda_{T_2,F}}{\lambda_{T_1,F}+\lambda_{T_2,F}}, \qquad \omega_{T_2,F} = \tfrac{\lambda_{T_1,F}}{\lambda_{T_1,F}+\lambda_{T_2,F}},$$

where $\lambda_{T_i,F} := \boldsymbol{\kappa}_{|T_i}\mathbf{n}_F\cdot\mathbf{n}_F$ for $i \in \{1, 2\}$. Set, for all $(u_h, v_h) \in V_h^{\mathrm{ccg}} \times V_h^{\mathrm{ccg}}$,

$$\begin{aligned} a_h^{\mathrm{ccg}}(u_h, v_h) := \int_\Omega \boldsymbol{\kappa}\nabla_h u_h\cdot\nabla_h v_h - \sum_{F\in\mathscr{F}_h} \int_F \left[\{\!\{\boldsymbol{\kappa}\nabla_h u_h\}\!\}_\omega\cdot\mathbf{n}_F[\![v_h]\!] + [\![u_h]\!]\{\!\{\boldsymbol{\kappa}\nabla v_h\}\!\}_\omega\cdot\mathbf{n}_F\right] \\ + \sum_{F\in\mathscr{F}_h} \eta\frac{\gamma_F}{h_F}\int_F [\![u_h]\!][\![v_h]\!], \end{aligned} \tag{12}$$

with ∇_h broken gradient on $\mathscr{T}_h$, $\gamma_F = \frac{2\lambda_{T_1,F}\lambda_{T_2,F}}{\lambda_{T_1,F}+\lambda_{T_2,F}}$ on internal faces $F \subset \partial T_1 \cap \partial T_2$ and $\gamma_F = \boldsymbol{\kappa}_{|T}\mathbf{n}_F{\cdot}\mathbf{n}_F$ on boundary faces $F \subset \partial T \cap \partial\Omega$. The user-dependent parameter η should be chosen large enough to ensure stability. The CCG method reads

$$\text{Find } u_h \in V_h^{\text{ccg}} \text{ s.t. } a_h^{\text{ccg}}(u_h, v_h) = \int_\Omega f v_h \text{ for all } v_h \in V_h^{\text{ccg}}. \tag{13}$$

The bilinear form a_h^{ccg} has been originally introduced by Di Pietro, Ern and Guermond [12] in the context of dG methods for degenerate advection-diffusion-reaction problems. For $\boldsymbol{\kappa} = \mathbf{1}_d$, the bilinear form a_h^{ccg} becomes

$$\begin{aligned} a_h^{\text{sip}}(u_h, v_h) = \int_\Omega \nabla_h u_h{\cdot}\nabla_h v_h - \sum_{F\in\mathscr{F}_h}\int_F [\{\!\{\nabla_h u_h\}\!\}{\cdot}\mathbf{n}_F [\![v_h]\!] + [\![u_h]\!]\{\!\{\nabla_h v_h\}\!\}{\cdot}\mathbf{n}_F] \\ + \sum_{F\in\mathscr{F}_h}\frac{\eta}{h_F}\int_F [\![u_h]\!][\![v_h]\!], \end{aligned} \tag{14}$$

and a_h^{sip} is the bilinear form yielding the Symmetric Interior Penalty (SIP) method of Arnold [4]. For further details on the link between CCG and discontinuous Galerkin methods we refer to [8–10].

A hybrid finite volume method As a last example we consider a variant of the SUSHI scheme of [17]; see also [14] for a discussion on the link with the MFD methods of [5, 6]. This method is based on the gradient reconstruction (10), but stabilization is achieved in a rather different manner with respect to (12). More precisely, we define the linear residual operator $\mathfrak{r}_h : \mathbb{T}_h \times \mathbb{F}_h \to \mathbb{P}_d^0(\mathscr{P}_h)$ as follows: For all $T \in \mathscr{T}_h$ and all $F \in \mathscr{F}_T$,

$$\mathfrak{r}_h(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}})_{|\mathscr{P}_{T,F}} = \frac{d^{\frac{1}{2}}}{d_{T,F}}\left[v_F - v_T - \mathfrak{G}_h^{\text{green}}(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}})_{|T}{\cdot}(\mathbf{x}_F - \mathbf{x}_T)\right].$$

We observe, in passing, that the factor $d^{\frac{1}{2}}$ can in general be replaced by a user-defined stabilization parameter $\eta > 0$. The advantage of taking $\eta = d^{\frac{1}{2}}$ is that it yields the classical two-point method on $\boldsymbol{\kappa}$-orthogonal meshes. The discrete space for SUSHI method with hybrid unknowns is obtained as follows: (i) let $\mathscr{S}_h = \mathscr{P}_h$ and $\mathbb{V}_h = \mathbb{T}_h \times \mathbb{F}_h$; (ii) let $\mathfrak{G}_h = \mathfrak{G}_h^{\text{hyb}}$ with $\mathfrak{G}_h^{\text{hyb}}$ such that, for all $(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}}) \in \mathbb{T}_h \times \mathbb{F}_h$, all $T \in \mathscr{T}_h$ and all $F \in \mathscr{F}_T$,

$$\mathfrak{G}_h^{\text{hyb}}(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}})_{|\mathscr{P}_{T,F}} = \mathfrak{G}_h^{\text{green}}(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}})_{|T} + \mathfrak{r}_h(\mathbf{v}_h^{\mathscr{T}}, \mathbf{v}_h^{\mathscr{F}})_{|\mathscr{P}_{T,F}}\mathbf{n}_{T,F}. \tag{15}$$

Denote by $\mathfrak{R}_h^{\text{hyb}}$ the reconstruction operator defined by (5) with $\mathfrak{G}_h = \mathfrak{G}_h^{\text{hyb}}$. The SUSHI method with hybrid unknowns reads

$$\text{Find } u_h \in V_h^{\text{hyb}} \text{ s.t. } a_h^{\text{sushi}}(u_h, v_h) = \int_\Omega f v_h \text{ for all } v_h \in V_h^{\text{hyb}},$$

with $a_h^{\text{sushi}}(u_h, v_h) := \int_\Omega \boldsymbol{\kappa} \nabla_h u_h \cdot \nabla_h v_h$ and ∇_h broken gradient on $\mathscr{P}_h$. Alternatively, one can obtain a cell centered version by setting $\mathbb{V}_h = \mathbb{T}_h$ and replacing $\mathfrak{G}_h^{\text{hyb}}$ defined by (15) by $\mathfrak{G}_h = \mathfrak{G}_h^{\text{cc}}$ with $\mathfrak{G}_h^{\text{cc}}$ such that

$$\forall \mathbf{v}_h \in \mathbb{T}_h, \qquad \mathfrak{G}_h^{\text{cc}}(\mathbf{v}_h) = \mathfrak{G}_h^{\text{hyb}}(\mathbf{v}_h, \mathbf{T}_h^{\text{g}}(\mathbf{v}_h)), \tag{16}$$

and trace operator $\mathbf{T}_h^{\text{g}}$ defined by (9). This variant coincides with the version proposed in [17] for homogeneous $\boldsymbol{\kappa}$, but it has the advantage to reproduce piecewise affine solutions of (7) on $\mathscr{T}_h$ when $\boldsymbol{\kappa}$ is heterogeneous. The discrete space obtained taking $\mathfrak{G}_h = \mathfrak{G}_h^{\text{cc}}$ in (6) is labeled V_h^{cc}.

3 Implementation

The goal of this section is to lay the foundations for a DSL embedded in the C++ language which transposes the mathematical concepts of §2 into practical implementations. To illustrate the capabilities of the DSL in a nutshell, compare Listing 1 with the expression of the bilinear form a_h^{sip} (14). The material is organized as follows: §3.1 introduces the algebraic back-end aiming at replacing the table of DOFs in the context of a element-like assembly procedure; §3.2 deals with more abstract concepts that mimic function spaces, linear and bilinear forms to offer a functional front-end.

3.1 Algebraic back-end

In this section we focus on the elementary ingredients to build the terms appearing in the linear and bilinear forms of §2, which constitute the back-end of the DSL presented in §3.2.

Linear combination The point of view presented in §2 naturally leads to finite element-like assembly of local contributions stemming from integrals over elements or faces. However, a few major differences have to be taken into account: (i) the stencil of the local contributions may vary from term to term; (ii) the stencil may be data-dependent, as is the case for the methods of §2 based on the L-construction; (iii) the stencil may be non-local, as DOFs from neighboring elements may enter in local reconstructions. All of the above facts invalidate the classical approach based on a global table of DOFs inferred from a mesh and a finite element in the sense of Ciarlet. Our approach to meet the above requirements is to (i) drop the concept

Listing 1 Implementation of the bilinear form a_h^{sip} defined by (14) using the DSL of §3

```
// Define discrete spaces, test and trial functions; c.f. Table 1
typedef FunctionSpace<span<Polynomial<d, 1> >,
                      gradient<GreenFormula<LInterpolator> >
                      >::type CCGSpace;
CCGSpace Vh(𝒯h);
Vh.gradientReconstruction().trace().set(DiffusionCoefficient, κ);
CCGSpace::TrialFunction uh(Vh, "uh");
CCGSpace::TestFunction  vh(Vh, "vh");
// Define the bilinear form
Form2 ah =
  integrate(All<Cell>::items(𝒯h), dot(grad(uh),grad(vh)))
 -integrate(All<Face>::items(𝒯h), dot(N(),avg(grad(uh)))*jump(vh)
                                  +dot(N(),avg(grad(vh)))*jump(uh))
 +integrate(All<Face>::items(𝒯h), η/H()*jump(uh)*jump(vh));
// Evaluate the bilinear form
MatrixContext context(A);
evaluate(ah, context);
```

of local element, and to refer to DOFs by a unique global index; (ii) introduce the concept of `LinearCombination` (with template parameters to be specified in what follows), which realizes a linear application from $\mathbb{V}_h$ onto the space $\mathbb{T}_r$ of real tensors of order $r \leq 2$.

In practice, a `LinearCombination` is an efficient mapping of the DOFs in $\mathbb{V}_h$ onto the corresponding coefficients in $\mathbb{T}_r$. A `LinearCombination` $\mathtt{l}^r$ can indeed be thought of as a list of couples $(I, \tau_{\mathtt{l},I})_{I\in\mathbb{I}_\mathtt{l}}$ where $\mathbb{I}_\mathtt{l} \subset \mathbb{V}_h$ is the stencil (i.e., a vector of global DOFs) and $\tau_{\mathtt{l},I} \in \mathbb{T}_r$, $I \in \mathbb{I}_\mathtt{l}$, are the corresponding coefficients. To account for strongly enforced boundary conditions, `LinearCombination` also contains a constant coefficient $\tau_{\mathtt{l},0}$, so that the evaluation at $\mathbf{v}_h \in \mathbb{V}_h$ (obtained by calling the function `LinearCombination.eval(`$\mathbf{v}_h$`)`) actually returns

$$\mathtt{l}^r(\mathbf{v}_h) = \sum_{I\in\mathbb{I}_\mathtt{l}} \tau_{\mathtt{l},I} v_I + \tau_{\mathtt{l},0} \in \mathbb{T}_r.$$

It is useful to define efficient operations such as the sum and subtraction of linear combinations, as well as different kinds of products by constants. This allows, e.g., to implement the gradient $\mathfrak{G}_h^{\mathrm{green}}$ defined by (10) as described in line 6 of Listing 2. When needed, each DOF I can be represented as a `LinearCombination` containing only the couple $(I, 1)$. As a result, both the hybrid version with face unknowns (15) and the cell centered version (11) of the gradient reconstruction can be obtained from Listing 2 by simply changing the value returned by the trace interpolator $\mathbf{T}_h$`.eval(`F`)` in line 5. We also pinpoint that the tensor order is a template parameter of `LinearCombination` to reduce the need for dynamic allocation.

Listing 2 Implementation of the gradient reconstruction $\mathfrak{G}_h^{\text{green}}$ (10) for an element $T \in \mathcal{T}_h$. The gradient $\mathfrak{G}_h^{\text{ccg}}$ can be obtained by changing the value of the `LinearCombination` returned by $\mathbb{T}_h$ in line 5. Bufferization is used as a means to improve efficiency

```
LinearCombination<0> vT;
vT += LinearCombination<1>::Term($I_T$,1.);
LinearCombination<1, Buffer> buffer;
for($F \in \mathcal{F}_T$) {
  const LinearCombination<0> & vF = $\mathbf{T}_h$.eval($F$);
  buffer += $\frac{|F|_{d-1}}{|T|_d}$(vF-vT)$\mathbf{n}_{T,F}$;
}
LinearCombination<1, Vector> GT;
buffer.compact(GT);
```

In the implementation, particular care must be devoted to expressions containing the sum or subtraction of two linear combinations $\mathtt{l}_1^r$ and $\mathtt{l}_2^r$, since this involves computing the intersection of the corresponding set of DOFs, say $\mathbb{I}_{\mathtt{l}_1}$ and $\mathbb{I}_{\mathtt{l}_2}$ respectively. To overcome this difficulty, complicate expressions are computed in two steps: a first step in which duplicate DOF indices are allowed, followed by a compaction stage where the algebraic sums of the corresponding coefficients are performed. This is obtained by changing the value of the second template parameter of `LinearCombination`. Specifically, in `Buffer`-mode a `LinearCombination` efficiently supports adding terms and can appear in the left-hand side of an assignment operator, while `Vector`-mode (default) only allows to traverse its elements in a fixed order; c.f. lines 3 and 8 of Listing 2.

Linear and bilinear contributions Exploiting the concept of `LinearCombination`, it is possible to devise a unified treatment for local contributions stemming from integrals over elements or faces. We illustrate the main ideas using the an example: For a given $T \in \mathcal{T}_h$ and for $u_h, v_h \in V_h^{\text{ccg}}$, we consider the local contribution $\mathbf{A}_{\text{loc}}$ associated to the term

$$\int_T \boldsymbol{\kappa} \nabla_h u_h \cdot \nabla_h v_h .$$

For the sake of simplicity we focus on the case when the constant coefficient $\tau_{\mathtt{l},0}$ is zero (in the example, this corresponds to the homogeneous Dirichlet boundary condition in problem (7)). The key remark is that both $(\boldsymbol{\kappa}\nabla_h u_h)_{|T} = \boldsymbol{\kappa}_{|T}\nabla(u_{h|T})$ and $(\nabla_h v_h)_{|T} = \nabla(v_{h|T})$ can be represented as objects of type `LinearCombination<1>`, say $\mathtt{l}_u^1 = (J, \tau_{\mathtt{l}_u,J})_{J\in\mathbb{J}}$ and $\mathtt{l}_v^1 = (I, \tau_{\mathtt{l}_v,I})_{I\in\mathbb{I}}$. The associated local contribution reads

$$\mathbf{A}_{\text{loc}} = [|T|_d \tau_{\mathtt{l}_v,I} \cdot \tau_{\mathtt{l}_u,J}]_{I\in\mathbb{I},\, J\in\mathbb{J}} . \tag{17}$$

Generalizing the above remark, one can implement local terms in matrix assembly as `BilinearContributions` which can be represented as triplets $(\mathbb{I}, \mathbb{J}, \mathbf{A}_{\text{loc}})$

Listing 3 Assembly of a bilinear and linear contribution (**A** represents here the global matrix **b** the global right-hand side vector)

```
LinearCombination<r> l_u^r, l_v^r;
// Assemble a bilinear contribution into the left-hand side
BilinearContribution<r> blc(γ, l_u^r, l_v^r);
A << blc;
// Assemble a linear contribution into the right-hand side
LinearContribution<r> lc(γ, l_v^r);
b << lc;
```

containing two vectors of DOF indices $\mathbb{I}$ and $\mathbb{J}$ and the local matrix $\mathbf{A}_{\mathrm{loc}}$. Observe, in particular, that $\mathbb{I}$ and $\mathbb{J}$ play the same role as the lines of the table of DOFs corresponding to test and trial functions supported in T in standard finite element implementations. As such, they are related to the lines and columns of the global matrix $\mathbf{A}$ to which $\mathbf{A}_{\mathrm{loc}}$ contributes,

$$\mathbf{A}(\mathbb{I},\mathbb{J}) \leftarrow \mathbf{A}(\mathbb{I},\mathbb{J}) + \mathbf{A}_{\mathrm{loc}}. \tag{18}$$

When the `LinearCombinations` concurring to a local term take values in $\mathbb{T}_r$, the vector inner product in (17) should be replaced by the appropriate tensor contraction. The additional argument γ appearing in Listing 3 serves as a multiplicative factor for the whole expression (in the above example, $\gamma = |T|_d$). More generally, γ can be a function of space and time, and may depend on discrete variables.

Similarly, `LinearContributions` serve to represent right-hand side contributions. `LinearContributions` are not detailed here for the sake of brevity. A typical assembly pattern is described in Listing 3. In particular, line 4 is the programming counterpart of (18).

3.2 *Functional front-end*

A further level of abstraction can be reached defining a DSL that allows to conceal all technical details and provide only the relevant components in a form as close as possible to the mathematical formulations of §2. We focus here, in particular, on the programming equivalent of discrete spaces and bilinear forms.

Function spaces Incomplete broken polynomial spaces defined by (6) are mapped onto C++ types conforming to the `FunctionSpace` concept detailed in Listing 4. The actual types are generated by a helper template class parametrized by a containing polynomial space, labeled **`span`**, and a piecewise constant gradient reconstruction, labeled **`gradient`** (labels for template arguments are here defined using the `boost::parameter` library). An example of usage is provided in lines 2–4 in Listing 1. The gradient reconstruction implicitly fixes both the

Listing 4 `FunctionSpace` concept

```
class FunctionSpace {
  // Types for trial and test functions
  class TrialFunction;
  class TestFunction;
  // Constructor
  FunctionSpace(const Mesh &);
  // Constant value of 𝔊_{h|S} for S ∈ 𝒮_h as a vector-valued linear combination of DOFs
  const LinearCombination<1> & Gh(S) const;
  // Value of ℜ_{h|S}(x) for x ∈ S and S ∈ 𝒮_h as a scalar-valued linear combination of DOFs
  const LinearCombination<0> & Rh(S, x) const;
};
```

Table 1 **`span`** and **`gradient`** template parameters for the class `FunctionSpace` corresponding to the discrete spaces of §2

Space	$\mathscr{S}_h$	**`span`**	**`gradient`**
$\mathbb{P}^0_d(\mathscr{T}_h)$	$\mathscr{T}_h$	`Polynomial<d, 0>`	`Null`
V_h^{g}	$\mathscr{P}_h$	`Polynomial<d, 1>`	`GFormula`
V_h^{ccg}	$\mathscr{T}_h$	`Polynomial<d, 1>`	`GreenFormula<LInterpolator>`
V_h^{hyb}	$\mathscr{P}_h$	`Polynomial<d, 1>`	`SUSHIFormula<HybridUnknowns>`
V_h^{cc}	$\mathscr{P}_h$	`Polynomial<d, 1>`	`SUSHIFormula<LInterpolator>`

choice (6) for the space of DOFs and the choice (3) for $\mathscr{S}_h$. The programming counterparts of the function spaces appearing in §2 are listed in Table 1.

The key role of a `FunctionSpace` is to bridge the gap between the algebraic representation of DOFs and the functional representation used in the methods of §2. This is achieved by the functions `Gh` and `Rh`, which are the `C++` counterpart of the linear operators $\mathfrak{G}_h$ and $\mathfrak{R}_h$ respectively; see §2.1. More specifically, (i) for all $S \in \mathscr{S}_h$, `Gh(`S`)` returns a vector-valued linear combination corresponding to the (constant) restriction $\mathfrak{G}_{h|S}$; (ii) for all $S \in \mathscr{S}_h$ and all $\mathbf{x} \in S$, `Rh(`S`, `$\mathbf{x}$`)` returns a scalar-valued linear combination corresponding to $\mathfrak{R}_{h|S}(\mathbf{x})$ defined according to (5). The linear combinations returned by `Gh` and `Rh` can be used to generate `LinearContributions` and `BilinearContributions` to build linear and bilinear terms as described above. A `FunctionSpace` also defines the types `TestFunction` and `TrialFunction` that correspond to the mathematical notions of test and trial functions in variational formulations. The main difference between a `TestFunction` and a `TrialFunction` is that the latter is associated to a vector of DOFs which is stored in memory. In addition, when used to define bilinear contributions, test and trial functions are associated to the lines and columns of the local matrix respectively. We conclude by observing that the choice of identifying test and trial functions by their type is in contrast with the approach of [20, §3.4], where special keywords accomplish this task.

Linear and bilinear forms Linear and bilinear forms are obtained as sums of linear and bilinear terms resulting from the composition of `TestFunctions` and

Listing 5 CCG discretization of the Stokes problem

```
CCGSpace::VectorTrialFunction uh(d);
CCGSpace::VectorTestFunction vh(d);
P0Space::TrialFunction ph;
P0Space::TestFunction qh;
Range::Index i(Range(0,dim-1));
Form2 ah, bh, sh;
ah = integrate(All<Cell>::items(𝒯h),
               sum(i)(dot(grad(uh(i)),grad(vh(i))) ))
    +integrate(Internal<Face>::items(𝒯h),
               sum(i)(-dot(fn,avg(grad(uh(i))))*jump(vh(i))
                      -jump(uh(i))*dot(N(),avg(grad(vh(i))))
                      +η/H()*jump(uh(i))*jump(vh(i))));
bh =-integrate(Internal<Face>::items(𝒯h),
               jump(ph)*dot(N(),avg(vh)));
sh = integrate(Internal<Face>::items(𝒯h),H()*jump(ph)*jump(qh));
```

`TrialFunctions` (or unary modifications thereof) by suitable tensor contractions. Examples of tensor contractions in Listing 1 are the **dot** and `*` operators. Products by functions of space, time and possibly discrete variables are also allowed. In Listing 1 we also display examples of geometric operators such as **N**`()` and **H**`()`, which allow to access face normals and diameters respectively. Unary modifiers encountered in Listing 1 are **grad**, **avg** and **jump**, corresponding, respectively, to the broken gradient on $\mathscr{S}_h$ and to the average and jump operators defined by (4). When applied to a test or trial function, a unary modifier is an object capable of returning a `LinearCombination` at evaluation.

By default, linear and bilinear forms are represented by vectors and (sparse) matrices, but other representations are possible resulting, e.g., in matrix-free implementations. In contrast with [20], the expression corresponding to a linear (resp. bilinear) form is stored as a property of an object **Form1** (resp. **Form2**) instead of being evaluated on-the-fly. This allows, in particular, to change the operations actually performed at evaluation according to a context. Changing the representation of linear and bilinear forms thus amounts to changing the context of evaluation. An example of evaluation is provided in lines 16–17 of Listing 1, where the global matrix **A** is assembled according to the expression of `ah` and to the procedure defined in `MatrixContext`. During the evaluation, each term in the expression of `ah` generates a corresponding `BilinearContribution`, which is in turn assembled as described in §3.1.

To conclude, we present a more complicate example involving a system of PDEs. More specifically, we consider the Stokes problem:

$$-\triangle u + \nabla p = f \text{ in } \Omega, \quad \nabla{\cdot}u = 0 \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega,$$

with $\langle p\rangle_\Omega = 0$ to ensure well-posedness. Let $X_h := [V_h^{\mathrm{ccg}}]^d \times \mathbb{P}^0_d(\mathscr{T}_h)/\mathbb{R}$. In Listing 5 we present the implementation of the CCG method of [9, §3]: Find

$(u_h, p_h) \in X_h$ such that

$$a_h(u_h, v_h) + b_h(v_h, p_h) - b_h(u_h, q_h) + s_h(p_h, q_h) = \int_\Omega f \cdot v_h, \qquad \forall (v_h, q_h) \in X_h$$

where $a_h(u_h, v_h) := \sum_{i=1}^{d} a_h^{\mathrm{sip}}(u_{h,i}, v_{h,i})$, $b_h(p_h, v_h) := -\sum_{F \in \mathcal{F}_h^{\mathrm{i}}} \int_F [\![p_h]\!] \{\!\{v_h\}\!\} \cdot \mathbf{n}_F$ and $s_h(p_h, q_h) := \sum_{F \in \mathcal{F}_h^{\mathrm{i}}} h_F \int_F [\![p_h]\!][\![q_h]\!]$. Notice the use of the **`sum`** keyword.

Acknowledgements Fruitful discussions with Christophe Prud'homme (Laboratoire Jean Kuntzmann, University of Grenoble) are gratefully acknowledged. We also wish to thank all the contributors to the `Arcane` platform.

References

1. I. Aavatsmark, T. Barkve, Ø. Bøe, and T. Mannseth. Discretization on unstructured grids for inhomogeneous, anisotropic media, Part I: Derivation of the methods. *SIAM J. Sci. Comput.*, 19(5):1700–1716, 1998.
2. I. Aavatsmark, G. T. Eigestad, B. T. Mallison, and J. M. Nordbotten. A compact multipoint flux approximation method with improved robustness. *Numer. Methods Partial Differ. Eq.*, 24:1329–1360, 2008.
3. L. Agélas, D. A. Di Pietro, and J. Droniou. The G method for heterogeneous anisotropic diffusion on general meshes. *M2AN Math. Model. Numer. Anal.*, 44(4):597–625, 2010.
4. D. N. Arnold. An interior penalty finite element method with discontinuous elements. *SIAM J. Numer. Anal.*, 19:742–760, 1982.
5. F. Brezzi, K. Lipnikov, and M. Shashkov. Convergence of mimetic finite difference methods for diffusion problems on polyhedral meshes. *SIAM J. Numer. Anal.*, 43(5):1872–1896, 2005.
6. F. Brezzi, K. Lipnikov, and V. Simoncini. A family of mimetic finite difference methods on polygonal and polyhedral meshes. *M3AS*, 15:1533–1553, 2005.
7. I. Danaila, F. Hecht, and O. Pironneau. *Simulation numérique en C++*. Dunod, Paris, 2003. http://www.freefem.org.
8. D. A. Di Pietro. Cell centered Galerkin methods. *C. R. Acad. Sci. Paris, Ser. I*, 348:31–34, 2010.
9. D. A. Di Pietro. Cell centered Galerkin methods for diffusive problems. Submitted. Preprint available at http://hal.archives-ouvertes.fr/hal-00511125/en/, September 2010.
10. D. A. Di Pietro. A compact cell-centered Galerkin method with subgrid stabilization. *C. R. Acad. Sci. Paris, Ser. I.*, 348(1–2):93–98, 2011.
11. D. A. Di Pietro and A. Ern. *Mathematical aspects of discontinuous Galerkin methods*. Mathematics & Applications. Springer-Verlag, Berlin, 2010. To appear.
12. D. A. Di Pietro, A. Ern, and J.-L. Guermond. Discontinuous Galerkin methods for anisotropic semi-definite diffusion with advection. *SIAM J. Numer. Anal.*, 46(2):805–831, 2008.
13. J. Droniou and R. Eymard. A mixed finite volume scheme for anisotropic diffusion problems on any grid. *Num. Math.*, 105(1):35–71, 2006.
14. J. Droniou, R. Eymard, T. Gallouët, and R. Herbin. A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods. *M3AS, Math. Models Methods Appl. Sci.*, 20(2):265–295, 2010.
15. M. G. Edwards and C. F. Rogers. Finite volume discretization with imposed flux continuity for the general tensor pressure equation. *Comput. Geosci.*, 2:259–290, 1998.

16. A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*, volume 159 of *Applied Mathematical Sciences*. Springer-Verlag, New York, NY, 2004.
17. R. Eymard, Th. Gallouët, and R. Herbin. Discretization of heterogeneous and anisotropic diffusion problems on general nonconforming meshes SUSHI: a scheme using stabilization and hybrid interfaces. *IMA J. Numer. Anal.*, 4(30):1009–1043, 2010.
18. G. Grospellier and B. Lelandais. The Arcane development framework. In *Proceedings of the 8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, pages 4:1–4:11, New York, NY, USA, 2009. ACM.
19. A. Logg and G. N. Wells. DOLFIN: Automated finite element computing. *ACM TOMS*, 37, 2010.
20. C. Prud'homme. A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations. *Sci. Prog.*, 14(2):81–110, 2006.

The paper is in final form and no similar paper has been or is being submitted elsewhere.