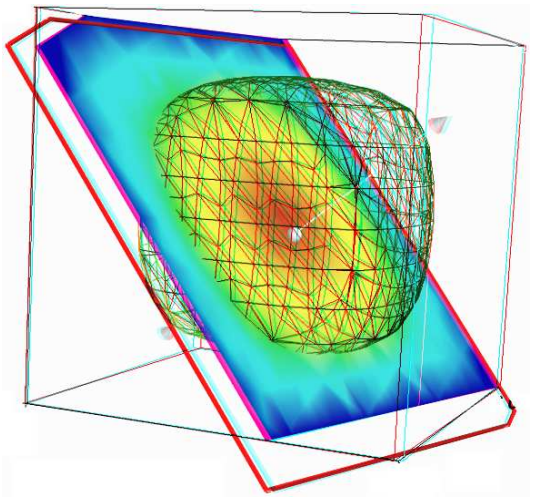
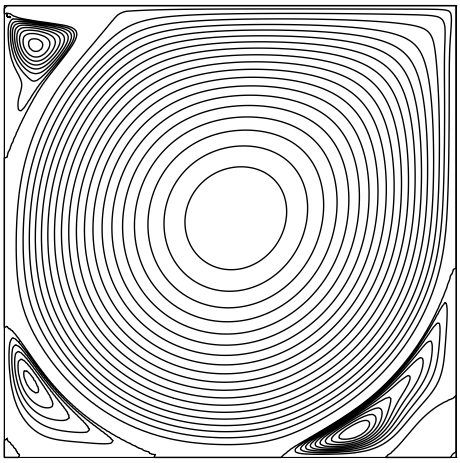
Efficient C++ finite element computing with Rheolef

HDG methods: in development stage

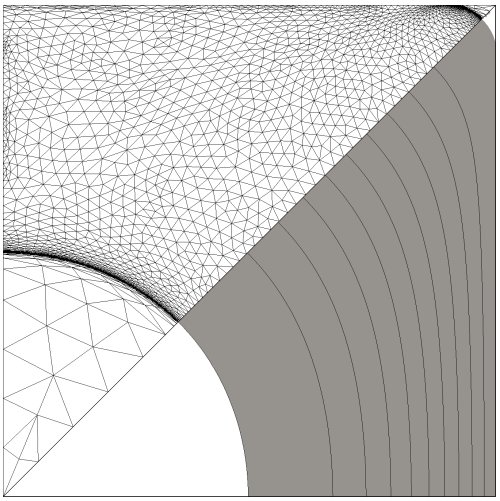
Pierre Saramito

version 7.1

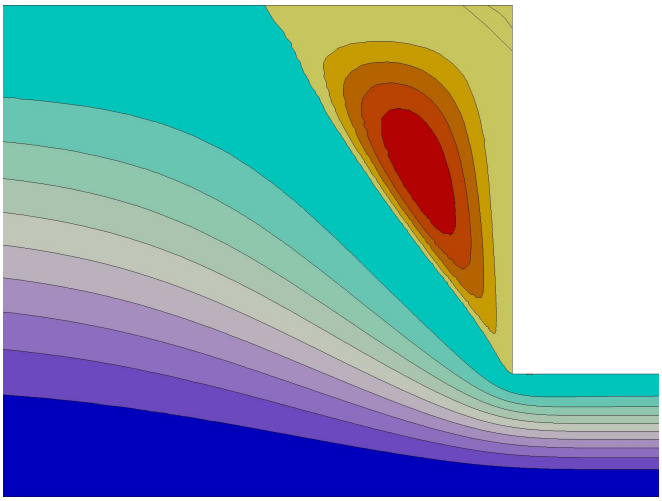




$Re = 10\,000$



$Bi = 0.5$



$We = 0.7$

Introduction

Rheolef is a programming environment for finite element method computing. The reader is assumed to be familiar with (i) the c++ programming language and (ii) the finite element method. As a Lego game, the Rheolef bricks allow the user to solve most problems, from simple to complex multi-physics ones, in few lines of code. The concision and readability of codes written with Rheolef is certainly a major keypoint of this environment. Here is an example of a Rheolef code for solving the Poisson problem with homogeneous boundary conditions:

Example: find u such that $-\Delta u = 1$ in Ω and $u = 0$ on $\partial\Omega$

```
int main (int argc, char** argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  Xh.block ("boundary");
  trial u (Xh); test v (Xh);
  form a = integrate (dot(grad(u),grad(v)));
  field lh = integrate (v);
  field uh (Xh);
  uh ["boundary"] = 0;
  problem p (a);
  p.solve (lh, uh);
  dout << uh;
}
```

Let $\Omega \subset \mathbb{R}^N, N = 1, 2, 3$

$X_h = \{v \in H^1(\Omega);\ v_{|K} \in P_k,\ \forall K \in \mathcal{T}_h\}$

$V_h = X_h \cap H_0^1(\Omega)$

$$a(u,v) = \int_\Omega \nabla u.\nabla v\, dx$$

$$l(v) = \int_\Omega v\, dx$$

(P) : find $u_h \in V_h$ such that

$$a(u_h, v_h) = l(v_h), \quad \forall v_h \in V_h$$

The right column shows the one-to-one line **correspondence between the code and the variational formulation**. Let us quote Stroustrup [2002], the conceptor of the c++ language:

> *"The time taken to write a program is at best roughly proportional to the* **number of lines** *written, and so is the number of errors in that code. If follows that a good way of writing correct programs is to write* **short programs**. *In other words, we need good libraries to allow us to write correct code that performs well. This in turn means that we need libraries to get our programs finished in a reasonable time. In many fields, such c++ libraries exist."*

Rheolef is an attempt to provide such a library in the field of finite element methods for partial differential equations. Rheolef provides both a c++ library and a set of unix commands for **shell** programming, providing data structures and algorithms [Wirth, 1985].

- Data structures fit the variational formulation concept: **field**, bilinear **form** and functional space, are c++ types for variables. They can be combined in algebraic expressions, as you write it on the paper.
- Algorithms refer to the most up-to-date ones: direct an iterative sparse matrix solvers for linear systems. They supports efficient distributed memory and parallel computations. Nonlinear c++ generic algorithms such as fixed point, damped Newton and continuation methods are also provided.

General *high order* piecewise polynomial finite element approximations are implemented, together with some mixed combinations for Stokes and incompressible elasticity. The *characteristic method* can be used for diffusion-convection problems while hyperbolic systems can be discretized by the discontinuous Galerkin method.

Contacts

email	Pierre.Saramito@imag.fr
home page	http://www-ljk.imag.fr/membres/Pierre.Saramito/rheolef

Please send all patches, comments and bug reports by mail to

rheolef@grenet.fr

Contents

Notations

Rheolef	mathematics	description
`d`	$d \in \{1,2,3\}$	dimension of the physical space
`interpolate(Vh,`*expr*`)`	$\pi_{V_h}(expr)$	interpolation in the space V_h
`integrate(omega,`*expr*`)`	$\int_\Omega expr \, \mathrm{d}x$	integration in $\Omega \subset \mathbb{R}^d$
`integrate(`omega, `on_local_sides(`*expr*`))`	$\sum_{K \in \mathscr{T}_h} \int_{\partial K} expr \, \mathrm{d}s$	integration on the local element sides
`dot(u,v)`	$\mathbf{u}.\mathbf{v} = \sum_{i=0}^{d-1} u_i v_i$	vector scalar product
`ddot(sigma,tau)`	$\sigma : \tau = \sum_{i,j=0}^{d-1} \sigma_{i,j} \tau_{i,j}$	tensor scalar product
`tr(sigma)`	$\mathrm{tr}(\sigma) = \sum_{i=0}^{d-1} \sigma_{i,i}$	trace of a tensor
`trans(sigma)`	σ^T	tensor transposition
`sqr(phi)` `norm2(phi)`	ϕ^2	square of a scalar
`norm2(u)`	$\lvert\mathbf{u}\rvert^2 = \sum_{i=0}^{d-1} u_i^2$	square of the vector norm
`norm2(sigma)`	$\lvert\sigma\rvert^2 = \sum_{i,j=0}^{d-1} \sigma_{i,j}^2$	square of the tensor norm
`abs(phi)` `norm(phi)`	$\lvert\phi\rvert$	absolute value of a scalar
`norm(u)`	$\lvert\mathbf{u}\rvert = \left(\sum_{i=0}^{d-1} u_i^2\right)^{1/2}$	vector norm
`norm(sigma)`	$\lvert\sigma\rvert = \left(\sum_{i,j=0}^{d-1} \sigma_{i,j}^2\right)^{1/2}$	tensor norm
`grad(phi)`	$\nabla\phi = \left(\frac{\partial\phi}{\partial x_i}\right)_{0 \leqslant i < d}$	gradient of a scalar field
`grad(u)`	$\nabla\mathbf{u} = \left(\frac{\partial u_i}{\partial x_j}\right)_{0 \leqslant i,j < d}$	gradient of a vector field
`div(u)`	$\mathrm{div}(\mathbf{u}) = \mathrm{tr}(\nabla\mathbf{u}) = \sum_{i=0}^{d-1} \frac{\partial u_i}{\partial x_i}$	divergence of a vector field
`D(u)`	$D(\mathbf{u}) = \left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)/2$	symmetric part of the gradient of a vector field
`curl(u)`	$\mathbf{curl}(\mathbf{u}) = \nabla \wedge \mathbf{u}$	curl of a vector field, when $d = 3$

Rheolef	mathematics	description
`curl(phi)`	$\mathbf{curl}(\phi) = \left(\frac{\partial\phi}{\partial x_1}, -\frac{\partial\phi}{\partial x_0}\right)$	curl of a scalar field, when $d=2$
`curl(u)`	$\mathrm{curl}(\mathbf{u}) = \frac{\partial u_1}{\partial x_0} - \frac{\partial u_0}{\partial x_1}$	curl of a vector field, when $d=2$
`grad_s(phi)`	$\nabla_s\phi = P\nabla\phi$ where $P = I - \mathbf{n}\otimes\mathbf{n}$	tangential gradient of a scalar
`grad_s(u)`	$\nabla_s\mathbf{u} = \nabla\mathbf{u}P$	tangential gradient of a vector
`Ds(u)`	$D_s(\mathbf{u}) = PD(\mathbf{u})P$	symmetrized tangential gradient
`div_s(u)`	$\mathrm{div}_s(\mathbf{u}) = \mathrm{tr}(D_s(\mathbf{u}))$	tangential divergence
`normal()`	$\mathbf{n}$	unit outward normal on $\Gamma = \partial\Omega$ or on an oriented surface Ω or on an internal oriented side S
`jump(phi)`	$[\![\phi]\!] = \phi_{\|K_0} - \phi_{\|K_1}$	jump accros inter-element side $S = \partial K_0 \cap K_1$
`average(phi)`	$\{\!\{\phi\}\!\} = (\phi_{\|K_0} + \phi_{\|K_1})/2$	average across S
`inner(phi)`	$\phi_{\|K_0}$	inner trace on S
`outer(phi)`	$\phi_{\|K_1}$	outer trace on S
`h_local()`	$h_K = \mathrm{meas}(K)^{1/d}$	length scale on an element K
`penalty()`	$\varpi_s = \max\left(\frac{\mathrm{meas}(\partial K_0)}{\mathrm{meas}(K_0)}, \frac{\mathrm{meas}(\partial K_1)}{\mathrm{meas}(K_1)}\right)$	penalty coefficient on S
`grad_h(phi)`	$(\nabla_h\phi)_{\|K} = \nabla(\phi_{\|K}), \forall K \in \mathcal{T}_h$	broken gradient
`div_h(u)`	$(\mathrm{div}_h\mathbf{u})_{\|K} = \mathrm{div}(\mathbf{u}_{\|K}), \forall K \in \mathcal{T}_h$	broken divergence of a vector field
`Dh(u)`	$(D_h(\mathbf{u}))_{\|K} = D(\mathbf{u}_{\|K}), \forall K \in \mathcal{T}_h$	broken symmetric part of the gradient of a vector field
`sin(phi)`	$\sin(\phi)$	standard mathematical functions extended to scalar fields
`cos(phi)`	$\cos(\phi)$	
`tan(phi)`	$\tan(\phi)$	
`acos(phi)`	$\cos^{-1}(\phi)$	
`asin(phi)`	$\sin^{-1}(\phi)$	
`atan(phi)`	$\tan^{-1}(\phi)$	
`cosh(phi)`	$\cosh(\phi)$	
`sinh(phi)`	$\sinh(\phi)$	

Rheolef	mathematics	description
`tanh(phi)`	$\tanh(\phi)$	
`exp(phi)`	$\exp(\phi)$	
`log(phi)`	$\log(\phi)$	
`log10(phi)`	$\log 10(\phi)$	
`floor(phi)`	$\lfloor\phi\rfloor$	largest integral not greater than ϕ
`ceil(phi)`	$\lceil\phi\rceil$	smallest integral not less than ϕ
`min(phi,psi)`	$\min(\phi,\psi)$	
`max(phi,psi)`	$\max(\phi,\psi)$	
`pow(phi,psi)`	ϕ^ψ	
`atan2(phi,psi)`	$\tan^{-1}(\psi/\phi)$	
`fmod(phi,psi)`	$\phi - \lfloor\phi/\psi + 1/2\rfloor\,\psi$	floating point remainder
`compose(f,phi)`	$f \circ \phi = f(\phi)$	applies an unary function f
`compose(f,phi1,...,phin)`	$f(\phi_1,\ldots,\phi_n)$	applies a n-ary function f, $n \geqslant 1$
`compose(phi,X)`	$\phi \circ X$, $X(x) = x + \mathbf{d}(x)$	composition with a characteristic

Chapter 1

[New] Approximation of the $H(\mathrm{div})$ space

1.1 The Raviart-Thomas element

The aim of this chapter is to introduce to the Raviart-Thomas element Raviart and Thomas [1977] for building an approximation of the $H(\mathrm{div}, \Omega)$ space.

There is a subtle issue. The **Rheolef** implementation choice for this element bases on internal interpolation nodes instead of moments. This choice leads to more efficient computation of degrees of freedom, but the standard Lagrange interpolation π_h no more satisfies the comutation diagram and optimal error in the $H(\mathrm{div}, \Omega)$ norm. Instead of the Lagrange interpolation π_h, we propose a projection operator, that satisfies these desired properties. We start building this projection operator for a piecewise discontinuous version of this element: it allows one to build a projection that requires only local operations and converges optimaly. Moreover, the piecewise discontinuous Raviart-Thomas approximation is used during the post-processing stage of the hybrid discontinuous Galekin method, that will be developped in a forthcoming chapter.

TODO: it remains to merge degreees of freedom along sides for obtaining a projector for the continuous Raviart-Thomas approximation, and check that the obtained projection still satisfies the commutation diagram and converges optimaly in $H(\mathrm{div}, \Omega)$. Indeed, even with a $C^1(\bar{\Omega})$ function, it is not clear that the obtained projection have degrees of freedom that matches along internal sides.

Let

$$\begin{aligned} V_h &= \left\{ \boldsymbol{v}_h \in \left(L^2(\Omega)\right)^d ; \;\; \boldsymbol{v}_{h|K} \in RT_k(K), \; \forall K \in \mathcal{T}_h \right\} \\ Q_h &= \left\{ q_h \in L^2(\Omega) \; ; \;\; q_{h|K} \in P_k, \; \forall K \in \mathcal{T}_h \right\} \end{aligned}$$

Here, V_h represents the space of discontinuous and piecewise k-th order Raviart-Thomas RT_k vector-valued functions while Q_h is the space of piecewise discontinuous polynomials.

Let π_{Q_h} denote the L_2 projection from $L^2(\Omega)$ into Q_h. For all $p \in L^2(\Omega)$, it is defined as $\pi_{Q_h}(p) = p_h \in Q_h$, where q_h is the solution of the following quadratic minimization problem:

$$p_h = \inf_{q_h \in Q_h} \int_\Omega (p - q_h)^2 \mathrm{d}x$$

Its solution is characterized as the unique solution of the following linear system, expressed in variational form:

(P_1): *find* $p_h \in Q_h$ *such that*

$$\int_\Omega p_h \, q_h \, \mathrm{d}x = \int_\Omega p \, q_h \, \mathrm{d}x, \quad \forall q_h \in Q_h$$

Following Roberts and Thomas [Roberts and Thomas, 1991, p. 551-552], our aim is to define π_{V_h} as the L_2 projection from $H(\mathrm{div},\Omega)$ into V_h and satisfying the following *commuting* property:

$$\begin{array}{ccc} H(\mathrm{div},\Omega) & \xrightarrow{\mathrm{div}} & L^2(\Omega) \\ \downarrow{\scriptstyle \pi_{V_h}} & & \downarrow{\scriptstyle \pi_{Q_h}} \\ V_h & \xrightarrow{\mathrm{div}} & Q_h \end{array}$$

i.e.

$$\mathrm{div}(\pi_{V_h}(\boldsymbol{u})) \quad = \quad \pi_{Q_h}(\mathrm{div}\,\boldsymbol{u}), \quad \forall \boldsymbol{u} \in H(\mathrm{div},\Omega) \tag{1.1a}$$

In [Roberts and Thomas, 1991, p. 553], theorem 6.3, this projection operator then satisfies an optimal error bound in the $H(\mathrm{div},\Omega)$ norm, i.e.:

$$\|u - \pi_{V_h}(\boldsymbol{u})\|_{0,2,\Omega} + \|\mathrm{div}(u - \pi_{V_h}(\boldsymbol{u}))\|_{0,2,\Omega} \quad = \quad \mathcal{O}(h^{k+1}) \tag{1.1b}$$

Remark that the Lagrange interpolation operator π_h from $H(\mathrm{div},\Omega)$ to V_h do not necessarily satisfy the commuting property (1.1a). Indeed, this depends upon the way the Raviart-Thomas internal degrees of freedom are chosen and implemented. When *internal degrees of freedom* are chosen as integrals over polynomials of degree $\ell \leqslant k-1$, e.g. as in [Roberts and Thomas, 1991, p. 551], eqn (6.8), then the Lagrange interpolation π_h satisfies both (1.1a) and (1.1b), as shown Roberts and Thomas [1991], theorem 6.1 and 6.3.

In practice, it is more efficient to choose for all the internal degrees of freedom of the Raviart-Thomas some values of the function on a set of internal points: this implementation choice has been chosen in **Rheolef**. In that case, the Lagrange interpolation π_h neither satisfies the commutation (1.1a) nor the bound (1.1b). More precisely, the Lagrange interpolation error is optimal in L^2 norm only while its divergence converges sub-optimally. Thus, with the present choice of the internal degrees of freedom, there is a need to explicitly compute the projection π_{V_h} that satisfies both (1.1a) and (1.1b).

For all $\boldsymbol{u} \in H(\mathrm{div},\Omega)$, this projection is defined as the L^2 projection of $\boldsymbol{u}$ under the constraint (1.1a):

$$\begin{array}{rcl} \pi_{V_h}(\boldsymbol{u}) & = & \displaystyle\inf_{\boldsymbol{v}_h \in V_h} \|\boldsymbol{u} - \boldsymbol{v}_h\|_{0,2,\Omega}^2 \\ & & \text{subject to} \quad \mathrm{div}(\boldsymbol{v}_v) \;=\; \pi_{Q_h}(\mathrm{div}\,\boldsymbol{u}) \end{array}$$

Then, $\boldsymbol{u}_h = \pi_{V_h}(\boldsymbol{u}) \in V_h$ is equivalently characterized as the solution of the following saddle-point problem:

$$(\boldsymbol{u}_h, p_h) \quad = \quad \inf_{\boldsymbol{v}_h \in V_h} \sup_{q_h \in Q_h} L(\boldsymbol{v}_h, q_h)$$

where the Lagrangian L is defined, for all $(\boldsymbol{v}_h, q_h) \in V_h \times Q_h$, by

$$L(\boldsymbol{v}_h, q_h) \quad = \quad \int_\Omega \left(|\boldsymbol{u} - \boldsymbol{v}_h|^2 + q_h \mathrm{div}(\boldsymbol{u} - \boldsymbol{v}_h) \right) \mathrm{d}x$$

The saddle-point of L is characterized as the unique solution of a linear system, expressed in variational form. Moreover, since both V_h and Q_h are spaces of piecewise discontinuous functions, the linear system writes as a collection of local linear systems on each element.

(P): *find* $(\boldsymbol{u}_h, p_h) \in V_h \times Q_h$ *such that, on each element* $K \in \mathcal{T}_h$, *we have*

$$\int_K (\boldsymbol{u}_h.\boldsymbol{v}_h + p_h \mathrm{div}\,\boldsymbol{v}_h)\mathrm{d}x \quad = \quad \int_K \boldsymbol{u}.\boldsymbol{v}_h \,\mathrm{d}x \tag{1.2a}$$

$$\int_K q_h \mathrm{div}\,\boldsymbol{u}_h \,\mathrm{d}x \quad = \quad \int_K q_h \mathrm{div}\,\boldsymbol{u} \,\mathrm{d}x \tag{1.2b}$$

for all $(\boldsymbol{v}_h, q_h) \in V_h \times Q_h$. Observe that q_h represents the Lagrange multiplier associated to the commutation constraint (1.2b), which is equivalent to (1.1a).

Let us introduce the following forms:

$$\begin{aligned} a(\boldsymbol{u}_h, p_h;\ \boldsymbol{v}_h, q_h) &= \int_\Omega (\boldsymbol{u}_h.\boldsymbol{v}_h + p_h \mathrm{div}\, \boldsymbol{v}_h + q_h \mathrm{div}\, \boldsymbol{u}_h)\, \mathrm{d}x \\ \ell(\boldsymbol{v}_h, q_h) &= \int_\Omega (\boldsymbol{u}.\boldsymbol{v}_h + q_h \mathrm{div}\, \boldsymbol{u})\, \mathrm{d}x \end{aligned}$$

The previous problem writes equivalently in abstract form:

(P): *find* $(\boldsymbol{u}_h, p_h) \in V_h \times Q_h$ *such that*

$$a(\boldsymbol{u}_h, p_h;\ \boldsymbol{v}_h, q_h) = \ell(\boldsymbol{v}_h, q_h), \quad \forall (\boldsymbol{v}_h, q_h) \in V_h \times Q_h$$

Note that the matrix associated to the bilinear form a is symmetric and block-diagonal: it can thus be efficiently inverted on the fly at the element level during the assembly process. The following code implement this efficient approach.

Example file 1.1: `commute_rtd.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "cosinus_vector.h"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  size_t d = omega.dimension(),
         k = (argc > 2) ? atoi(argv[2]) : 0;
  space Vh (omega, "RT"+itos(k)+"d"),
        Lh (omega, "P" +itos(k)+"d"),
        Xh = Vh*Lh;
  trial x (Xh); test y (Xh);
  auto u = x[0], lambda = x[1];
  auto v = y[0], mu     = y[1];
  integrate_option iopt;
  iopt.invert = true;
  form  inv_a = integrate (dot(u,v) + div_h(v)*lambda + div_h(u)*mu, iopt);
  field lh = integrate (dot(u_exact(d),v) + div_u_exact(d)*mu),
        xh = inv_a*lh,
        p_Vh_u = xh[0],
        pi_h_u = interpolate(Vh,u_exact(d));
  dout << catchmark("p_Vh_u") << p_Vh_u
       << catchmark("pi_h_u") << pi_h_u;
}
```

How to run the program ?

```
make commute_rtd commute_rtd
mkgeo_grid -t 10 > square.geo ./commute_rtd square.geo | ./commute_rtd_error
```

The program '`commute_rt.cc`' compute both the projection $\pi_{V_h}(u)$ and the standard Lagrange interpolation $\pi_h(u)$, while '`commute_rtd_error.cc`' performs the computation of the corresponding errors. The file '`cosinus_vector.h`' furnishes the function used for the present test:

$$\boldsymbol{u}(x) = \begin{pmatrix} \cos(x_0 + 2x_1) \\ \sin(x_0 - 2x_1) \end{pmatrix}$$

These two last files are not listed here but are available in the **Rheolef** example directory. Observe on Fig. 1.1 that the error for the projection $\pi_{V_h}(u)$ and its divergence behave as $\mathcal{O}(h^{k+1})$, which is optimal. Conversely, the error for the Lagrange interpolation $\pi_h(u)$ is sub-optimal for the divergence.

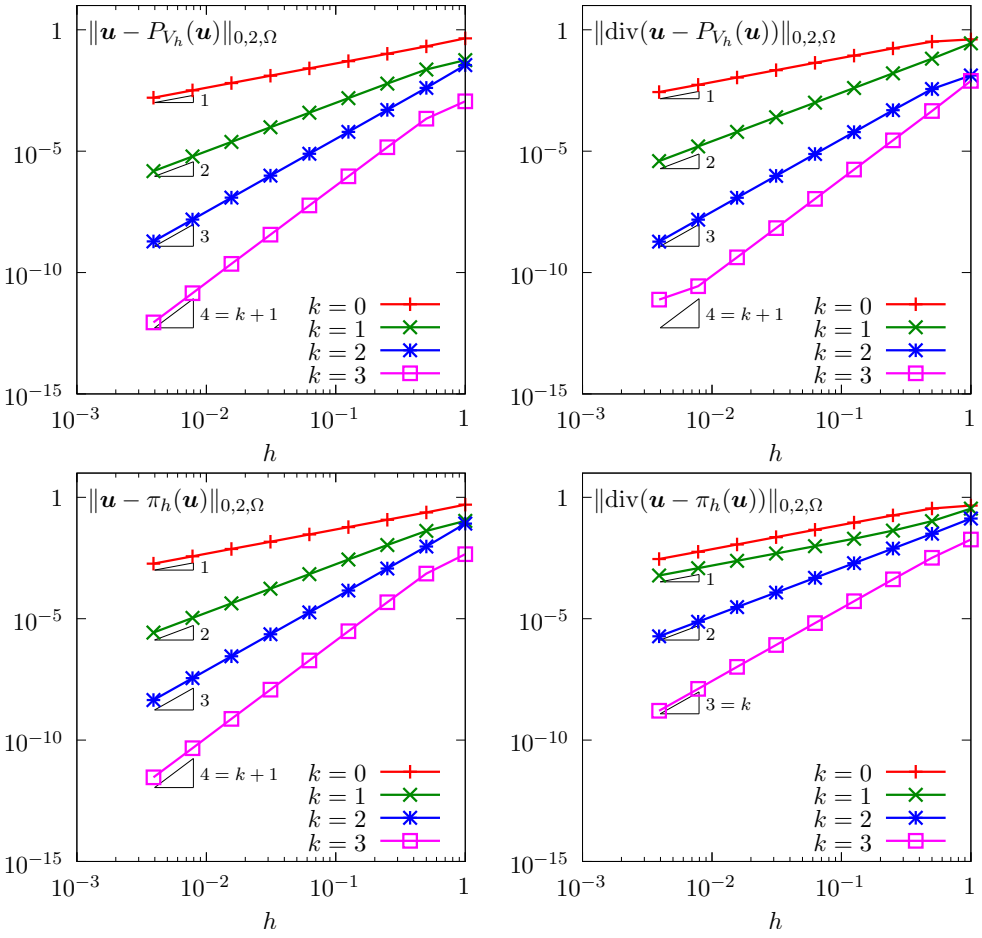


Figure 1.1: Raviart-Thomas approximation: π_{V_h} projection (top) and π_h interpolation (bottom) errors in L^2 norm for the approximation and its divergence.

In conclusion, the projection π_{V_h} should be used instead of the interpolation π_h when we want to build an optimal Raviart-Thomas approximation.

Chapter 2

[New] Hybrid discontinuous methods

2.1 Hybrid discontinuous Galerkin (HDG) methods

The aim of this chapter is to introduce to hybridization of discontinuous Galerkin methods within the **Rheolef** environment. For a review of hybridizable discontinuous Galerkin methods, see Nguyen et al. [2011]. The hybridization technique allows an efficient finite element implementation of many problems of importance, such as the Navier-Stokes one. Let us start by some model problems.

2.1.1 The Poisson problem

Let us consider the Poisson problem with mixed Dirichlet and Neumann boundary conditions:
(P): *find u, defined in Ω, such that*

$$\begin{array}{rcl} -\Delta u & = & f \ \text{ in } \ \Omega \\ u & = & g_d \ \text{ on } \ \Gamma_d \\ \dfrac{\partial u}{\partial n} & = & g_n \ \text{ on } \ \Gamma_n \end{array}$$

where $\partial\Omega = \Gamma_d \cup \Gamma_n$, and the interior of the two boundary domains Γ_d and Γ_n are disjoints. The data f, g_d and g_n are given. Let us introduce the gradient $\boldsymbol{\sigma} = \nabla u$ as an independent variable. The problem writes equivalently (see e.g. Nguyen et al. [2011]):
(P): *find $\boldsymbol{\sigma}$ and u, defined in Ω, such that*

$$\begin{cases} \boldsymbol{\sigma} - \nabla u = 0 \ \text{ in } \ \Omega & (2.1a) \\ \operatorname{div}(\boldsymbol{\sigma}) = -f \ \text{ in } \ \Omega & (2.1b) \\ u = g_d \ \text{ on } \ \Gamma_d & (2.1c) \\ \boldsymbol{\sigma}.\boldsymbol{n} = g_n \ \text{ on } \ \Gamma_n & (2.1d) \end{cases}$$

Let us multiply (2.1a) by a test function $\boldsymbol{\tau}$ and integrate by parts on any element K:

$$\int_K (\boldsymbol{\sigma}.\boldsymbol{\tau} + \operatorname{div}(\boldsymbol{\tau})\, u)\ \mathrm{d}x - \int_{\partial K} (\boldsymbol{\tau}.\boldsymbol{n})\, u\, \mathrm{d}s \quad = \quad 0$$

In the discontinuous Galerkin method, the trace of u on ∂K will be discontinuous across the inter-element boundaries. The hybridization replaces this trace by a new independent variable,

denoted as λ, that is only definied on sides of the mesh:

$$\int_K (\boldsymbol{\sigma}.\boldsymbol{\tau} + \mathrm{div}(\boldsymbol{\tau})\, u)\ \mathrm{d}x - \int_{\partial K} (\boldsymbol{\tau}.\boldsymbol{n})\, \lambda\, \mathrm{d}s \quad = \quad 0 \tag{2.2a}$$

The λ variable will be uni-valued on boundary inter-elements and accounts strongly for the boundary condition: $\lambda = g_d$ on Γ_d. Then, let us multiply (2.1b) by a test function v with a zero average value on K:

$$\int_K \mathrm{div}(\boldsymbol{\sigma})\, v\, \mathrm{d}x \quad = \quad -\int_K f\, v\, \mathrm{d}x$$

In order to weakly impose the condition $u = \lambda$ on ∂K, we add a penalization term:

$$\int_K \mathrm{div}(\boldsymbol{\sigma})\, v\, \mathrm{d}x - \beta h^n \int_{\partial K} h^n (u-\lambda)\, v\, \mathrm{d}s \quad = \quad -\int_K f\, v\, \mathrm{d}x \tag{2.2b}$$

Here, h denotes the local mesh size in the element K. The two constants $\beta > 0$ and $n \in \mathbb{R}$ are respectively a penalization coefficient and power index. Following Cockburn et al. [2009], we consider the three cases $n = 0$, 1 and -1. We have three unknowns $\boldsymbol{\sigma}$, u and λ and only the two equations (2.2a) and (2.2b). For the problem to be complete, we add an equation for λ. Observe that (2.2b) writes equivalently, after a second integration by part on K:

$$-\int_K \boldsymbol{\sigma}.\nabla v\, \mathrm{d}x + \int_{\partial K} (\boldsymbol{\sigma}.\boldsymbol{n} - \beta h^n (u-\lambda))\, v\, \mathrm{d}s \quad = \quad -\int_K f\, v\, \mathrm{d}x$$

The quantity involved in the integral on ∂K is denoted as $\widehat{\boldsymbol{\sigma}} = \boldsymbol{\sigma} - \beta h^n (u-\lambda)\, \boldsymbol{n}$ and refered as the *numerical flux* across inter-element boundaries. As an additional equation, we impose the continuity of the normal component of this numerical flux. On the Γ_n boundary domain, the normal component of the numerical flux is imposed to be the prescribed flux g_n, while the Dirichlet condition $\lambda = g_d$ is precribed on the Γ_d boundary domain:

$$\begin{aligned}
[\![\boldsymbol{\sigma} - \beta h^n (u-\lambda)\, \boldsymbol{n}]\!].\boldsymbol{n} &= 0 \ \text{ on } \ S,\ \forall S \in \mathscr{S}_h^{(i)} && (2.3a)\\
\boldsymbol{\sigma}.\boldsymbol{n} - \beta h^n (u-\lambda) &= g_n \ \text{ on } \ \Gamma_n && (2.3b)\\
\lambda &= g_d \ \text{ on } \ \Gamma_d && (2.3c)
\end{aligned}$$

For an internal side $S = \partial K_+ \cap \partial K_- \in \mathscr{S}_h^{(i)}$ between two elements $K_1, K_2 \in \mathscr{T}_h$, relation (2.3a) writes:

$$\boldsymbol{\sigma}_+.\boldsymbol{n}_+ - \beta h^n (u_+ - \lambda) + \boldsymbol{\sigma}_-.\boldsymbol{n}_- - \beta h^n (u_- - \lambda) \quad = \quad 0 \ \text{ on } \ S$$

where $\boldsymbol{\sigma}_\pm$ and $u_\pm$ are the trace on S of $\boldsymbol{\sigma}$ and u in $K_\pm$ and $\boldsymbol{n}_\pm$ are the outer normal of $K_\pm$ on S. Since S is oriented, let us choose without loss of generality $\boldsymbol{n} = \boldsymbol{n}_- = -\boldsymbol{n}_+$. Then, the previous relation writes:

$$\begin{aligned}
& (\boldsymbol{\sigma}_- - \boldsymbol{\sigma}_+).\boldsymbol{n} - \beta h^n (u_- + u_+) + 2\beta h^n \lambda \ = \ 0 \ \text{ on } \ S\\
\iff \quad & [\![\boldsymbol{\sigma}]\!].\boldsymbol{n} - 2\beta h^n (\{\!\{u\}\!\} - \lambda) \ = \ 0 \ \text{ on } \ S
\end{aligned}$$

where we have used the jump and average across the internal side S. The previous relation, together with (2.3c) and (2.3c), writes in variational form:

$$\int_{\mathscr{S}_h^{(i)}} ([\![\boldsymbol{\sigma}]\!].\boldsymbol{n} - 2\beta h^n (\{\!\{u\}\!\} - \lambda))\ \mu\, \mathrm{d}s + \int_{\partial\Omega} (\boldsymbol{\sigma}.\boldsymbol{n} - \beta h^n (u-\lambda))\ \mu\, \mathrm{d}s \quad = \quad \int_{\Gamma_n} g_n\, \mu\, \mathrm{d}s \tag{2.4}$$

for all test function μ, defined on all internal sides of $\mathscr{S}_h^{(i)}$ and on all sides of the boundary domain Γ_n and that vanishes on all sides of the boundary domain Γ_d.

Grouping (2.2a), (2.2b) and (2.4), we obtain the discrete variational formulation:

$(FV)_h$: *find* $(\boldsymbol{\sigma}_h, u_h, \lambda_h) \in T_h \times X_h \times \Lambda_h(g_d)$ *such that*

$$\begin{aligned}
\int_\Omega \boldsymbol{\sigma}.\boldsymbol{\tau}\,\mathrm{d}x + \int_\Omega \mathrm{div}_h(\boldsymbol{\tau})\,u\,\mathrm{d}x - \int_{\mathscr{S}_h^{(i)}} (\llbracket\boldsymbol{\tau}\rrbracket.\boldsymbol{n})\,\lambda\,\mathrm{d}s - \int_{\partial\Omega} (\boldsymbol{\tau}.\boldsymbol{n})\,\lambda\,\mathrm{d}s &= 0 \\
\int_\Omega \mathrm{div}_h(\boldsymbol{\sigma})\,v\,\mathrm{d}x - \beta \sum_{K\in\mathscr{T}_h} \int_{\partial K} h^n u\,v\,\mathrm{d}s + \int_{\mathscr{S}_h^{(i)}} 2\beta h^n \{\!\{v\}\!\}\,\lambda\,\mathrm{d}s + \int_{\partial\Omega} \beta h^n v\,\lambda\,\mathrm{d}s &= -\int_\Omega f\,v\,\mathrm{d}x \\
\int_{\mathscr{S}_h^{(i)}} (2\beta h^n \{\!\{u\}\!\} - \llbracket\boldsymbol{\sigma}\rrbracket.\boldsymbol{n})\,\mu\,\mathrm{d}s + \int_{\partial\Omega} (\beta h^n u - \boldsymbol{\sigma}.\boldsymbol{n})\,\mu\,\mathrm{d}s - \int_{\mathscr{S}_h^{(i)}} 2\beta h^n \lambda\,\mu\,\mathrm{d}s - \int_{\partial\Omega} \beta h^n \lambda\,\mu\,\mathrm{d}s &= -\int_{\Gamma_n} g_n\,\mu\,\mathrm{d}s
\end{aligned}$$

for all $(\boldsymbol{\tau}_h, v_h, \tau_h) \in T_h \times X_h \times \Lambda_h(0)$, where

$$\begin{aligned}
T_h &= \left\{\boldsymbol{\tau} \in \left(L^2(\Omega)\right)^d \;;\; \boldsymbol{\tau}_{|K} \in (P_k)^d,\ \forall K \in \mathscr{T}_h\right\} \\
X_h &= \left\{v \in L^2(\Omega) \;;\; v_{|K} \in P_k,\ \forall K \in \mathscr{T}_h\right\} \\
M_h &= \left\{\mu \in L^2(\mathscr{S}_h) \;;\; \mu_{|S} \in P_k,\ \forall S \in \mathscr{S}_h\right\} \\
\Lambda_h(g_d) &= \left\{\mu \in M_h \;;\; \mu_{|S} = \pi_h(g_d),\ \forall S \subset \Gamma_d\right\}
\end{aligned}$$

and $k \geqslant 0$ is the polynomial degree. This is a symmetric system with a mixed structure. Let:

$$\begin{aligned}
a_h((\boldsymbol{\sigma}, u), (\boldsymbol{\tau}, v)) &= \int_\Omega (\boldsymbol{\sigma}.\boldsymbol{\tau} + \mathrm{div}_h(\boldsymbol{\tau})\,u + \mathrm{div}_h(\boldsymbol{\sigma})\,v)\,\mathrm{d}x - \beta \sum_{K\in\mathscr{T}_h} \int_{\partial K} h^n u\,v\,\mathrm{d}s \\
b_h((\boldsymbol{\tau}, v), \mu) &= \int_{\mathscr{S}_h^{(i)}} (-\llbracket\boldsymbol{\tau}\rrbracket.\boldsymbol{n} + 2\beta h^n \{\!\{v\}\!\})\,\mu\,\mathrm{d}s + \int_{\partial\Omega} (-\boldsymbol{\tau}.\boldsymbol{n} + \beta h^n \{\!\{v\}\!\})\,\mu\,\mathrm{d}s \\
c_h(\lambda, \mu) &= \int_{\mathscr{S}_h^{(i)}} 2\beta h^n \lambda\,\mu\,\mathrm{d}s + \int_{\partial\Omega} \beta h^n \lambda\,\mu\,\mathrm{d}s \\
\ell_h(\boldsymbol{\tau}, v) &= -\int_\Omega f\,v\,\mathrm{d}x \\
k_h(\mu) &= -\int_{\Gamma_n} g_n\,\mu\,\mathrm{d}s
\end{aligned}$$

Then, the variational formulation writes equivalently:

$(FV)_h$: *find* $(\boldsymbol{\sigma}_h, u_h, \lambda_h) \in T_h \times X_h \times \Lambda_h(g_d)$ *such that*

$$\begin{aligned}
a_h((\boldsymbol{\sigma}, u), (\boldsymbol{\tau}, v)) + b_h((\boldsymbol{\tau}, v), \lambda) &= \ell_h(\boldsymbol{\tau}, v) \\
b_h((\boldsymbol{\sigma}, u), \mu) - c_h(\lambda, \mu) &= k_h(\mu)
\end{aligned}$$

for all $(\boldsymbol{\tau}_h, v_h, \tau_h) \in T_h \times X_h \times \Lambda_h(0)$. Let $\chi_h = (\boldsymbol{\sigma}_h, u_h)$. This linear symmetric system admits the following matrix structure:

$$\begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} \boldsymbol{\chi}_h \\ \lambda_h \end{pmatrix} = \begin{pmatrix} F \\ G \end{pmatrix}$$

A carreful study shows that the A matrix has a block-diagonal structure at the element level and can be easily inverted on the fly during the assembly process. The matrix structure writes equivalently:

$$\begin{aligned}
&\iff \left\{\begin{array}{rcl} A\boldsymbol{\chi}_h + B^T\lambda_h &=& F \\ B\boldsymbol{\chi}_h - C\lambda_h &=& G \end{array}\right. \\
&\iff \left\{\begin{array}{l} \boldsymbol{\chi}_h = A^{-1}(F - B^T\lambda_h) \\ (C + BA^{-1}B^T)\lambda_h = BA^{-1}F - G \end{array}\right.
\end{aligned}$$

The second equation is solved first: the only remaining unknown is the Lagrange multiplier λ_h, in a linear system involving the Schur complement matrix $S = C + BA^{-1}B^T$. Then, the two variables $\boldsymbol{\chi}_h = (\boldsymbol{\sigma}_h, u_h)$ are simply obtained by a direct computation.

Example file 2.1: `dirichlet_hdg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "sinusprod_dirichlet.h"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  string approx = (argc > 2) ?      argv[2]   : "P1d";
  Float  n      = (argc > 3) ? atof(argv[3]) : 1;
  Float  beta   = (argc > 4) ? atof(argv[4]) : 1;
  space Th (omega, approx, "vector"),
        Xh (omega, approx),
        Yh = Th*Xh,
        Mh (omega["sides"], approx);
  Mh.block("boundary");
  space Wh(Mh.get_geo()["boundary"],approx);
  size_t d = omega.dimension();
  size_t k = Xh.degree();
  trial x(Yh), lambda(Mh);
  test  y(Yh), mu(Mh);
  auto sigma = x[0], u = x[1];
  auto tau   = y[0], v = y[1];
  integrate_option iopt;
  iopt.invert = true;
  auto coef = beta*pow(h_local(),n);
  form inv_a = integrate(dot(sigma,tau) + u*div_h(tau) + v*div_h(sigma)
                         - on_local_sides(coef*u*v), iopt);
  form b = integrate("internal_sides",
             (-dot(jump(sigma),normal()) + 2*coef*average(u))*mu)
         + integrate("boundary", (-dot(sigma,normal()) + coef*u)*mu);
  form c = integrate("internal_sides", 2*coef*lambda*mu)
         + integrate("boundary",          coef*lambda*mu);
  field lh = integrate (-f(d)*v);
  field kh(Mh,0), lambda_h(Mh,0);
  lambda_h ["boundary"] = interpolate (Wh, g(d));
  form s = c + b*inv_a*trans(b);
  field rh = b*(inv_a*lh) - kh;
  problem p (s);
  p.solve (rh, lambda_h);
  field xh = inv_a*(lh - b.trans_mult(lambda_h));
  dout << catchmark("n")      << n << endl
       << catchmark("beta")   << beta << endl
       << catchmark("u")      << xh[1]
       << catchmark("lambda") << lambda_h
       << catchmark("sigma")  << xh[0];
}
```

The right-hand-side f and the Dirichlet boundary condition g has been chosen such that the exact solution is given by:

$$u(x) = \prod_{i=0}^{d-1} \sin(\pi x_i)$$

The files '`sinusprod_dirichlet.h`', that defines the data f and g, and '`sinusprod_error_hdg.cc`', that compute the error in various norms, are not listed here but are available in the **Rheolef** example directory.

How to run the program

The compilation and run write:

```
make dirichlet_hdg
mkgeo_grid -t 10 > square.geo
./dirichlet_hdg square.geo P1d > square.field
field square.field -elevation
```

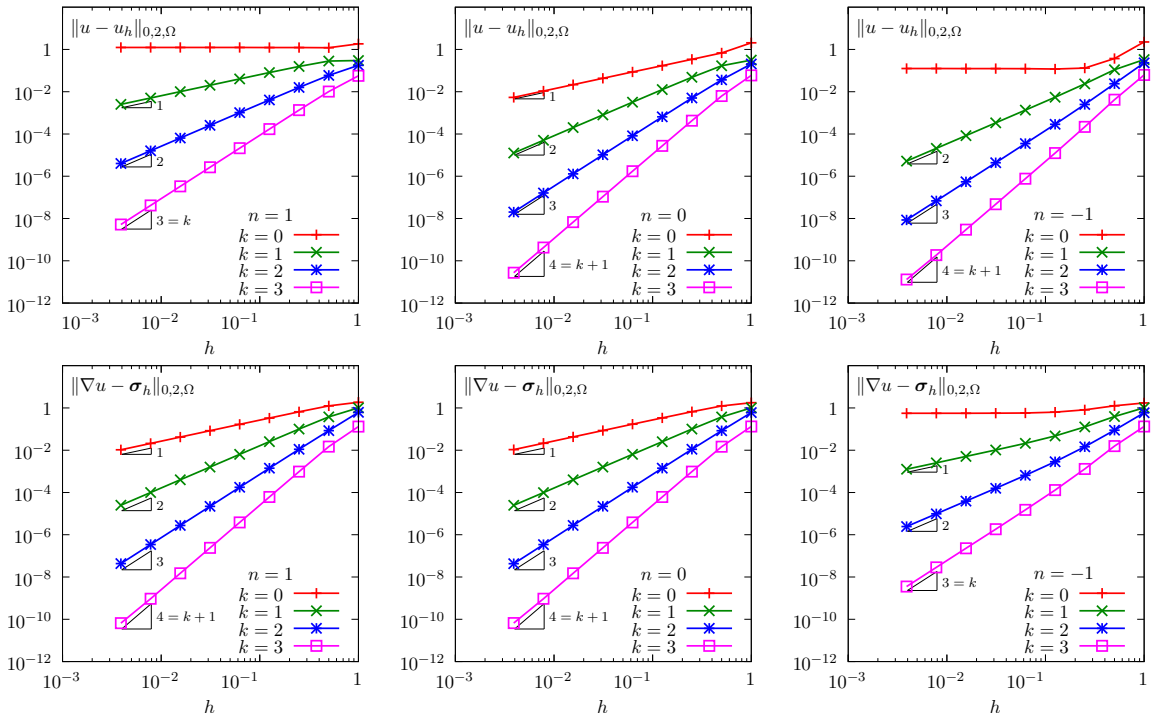


Figure 2.1: Hybrid discontinuous Galerkin method: Poisson problem with Dirichlet boundary conditions in 2D geometry. Convergence vs h and k for the approximation of the solution u_h and of its gradient $\boldsymbol{\sigma}_h$. (left) $n=1$; (center) $n=0$; (right) $n=-1$.

```
field square.field -elevation -mark lambda
field square.field -velocity  -mark sigma
make sinusprod_error_hdg
./sinusprod_error_hdg < square.field
```

Fig. 2.1 plots the errors vs h and k for a two dimensional geometry. Observe that the approximation u_h converges optimaly, as h^{k+1}, in the L^2 norm for any $k \geqslant 0$ when the power index $n=1$. When $n=1$, the convergence is suboptimal, as h^k only and note that the lowest order approximation $k=0$ is not convergent. When $n=-1$, the convergence is optimal, as h^{k+1}, only when $k \geqslant 1$, while the lowest order approximation $k=0$ is not convergent. The approximation $\boldsymbol{\sigma}_h$ of the gradient converges optimaly, as h^{k+1}, for both $n=0$ and 1, while it is suboptimal, as h^k, when $n=-1$. All these results are consistent with the approxiamtion theory of the HDG method, see Cockburn et al. [2009] and tables 2, 3, 6.

2.1.2 Superconvergence of the Lagrange multiplier

Let π_{M_h} denote the L_2 projection from $L^2(\mathscr{S}_h)$ into M_h. We consider the special case of computing the projection $\pi_{M_h}(u) \in M_h$ of the restriction to $\mathscr{S}_h$ of an element $u \in L^2(\mathscr{S}_h)$. It is defined as

$$\pi_{M_h}(u) \quad = \quad \underset{\mu_h \in M_h}{\arg\inf} \ \frac{1}{2} \sum_{S \in \mathscr{S}_h} \int_S (u-\mu_h)^2 \mathrm{d}s$$

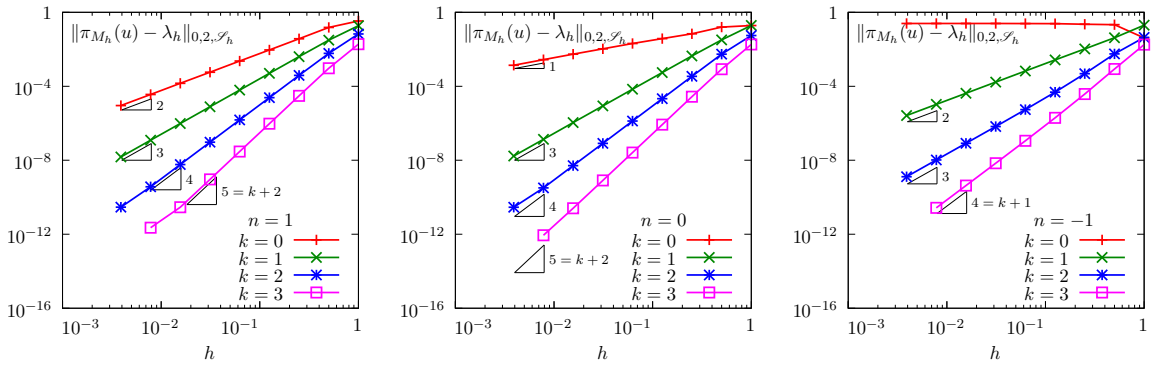


Figure 2.2: Hybrid discontinuous Galerkin method: Poisson problem with Dirichlet boundary conditions in 2D geometry. Super-convergence vs h and k for the Lagrange multiplier λ_h to the L^2 projection on M_h of the exact solution. (left) $n=1$; (center) $n=0$; (right) $n=-1$.

The following bilinear forms are introduced:

$$\begin{aligned} m_s(\lambda,\mu) &= \sum_{S\in\mathscr{S}_h} \int_S \lambda\,\mu\,\mathrm{d}s \\ k_h(\mu) &= \sum_{S\in\mathscr{S}_h} \int_S u\,\mu\,\mathrm{d}s \end{aligned}$$

Then, the projection reduces to:
find $\bar{\lambda}_h = \pi_{M_h}(u) \in M_h$ *such that*

$$m_s(\bar{\lambda}_h,\mu_h) = k_h(\mu_h), \quad \forall \mu_h \in M_h$$

The result of projection operator can be obtained by a resolution of a linear system. Following [Cockburn et al., 2008, p. 1896], in order to measure the error on the set of sides $\mathscr{S}_h$ of the mesh, we introduce the mesh-dependent norm $\|.\|_{0,2,\mathscr{S}_h}$, defined for all $\mu \in L^2(\mathscr{S}_h)$ by

$$\|\mu\|^2_{0,2,\mathscr{S}_h} = \sum_{S\in\mathscr{S}_h} \int_S h_S \mu^2\,\mathrm{d}s$$

where h_S is a characteristic length on the side $S \in \mathscr{S}_h$. Fig. 2.2 plots the difference $\pi_{M_h}(u) - \lambda_h$ in this mesh-dependent norm. In agrement with the theoretical results [Cockburn et al., 2008, p. 1896], we observe the superconvergence of the multiplier λ_h to the L^2 projection $\pi_{M_h}(u)$. More precisely, when $n = 1$ the order of convergence is $k+2$ for any $k \geqslant 0$. When $n = 0$, the superconvergence occurs only when $k \geqslant 1$ and when $n = -1$ there is no superconvergence. This error is also computed by the code '`sinusprod_error_hdg.cc`'.

2.1.3 Superconvergence of the piecewise averaged solution

Example file 2.2: `dirichlet_hdg_average.icc`

```
field dirichlet_hdg_average (field uh, field lambda_h) {
  size_t k = uh.get_space().degree();
  size_t d = uh.get_geo().dimension();
  space Zh (uh.get_geo(), "P0");
  trial zeta(Zh); test xi(Zh);
  integrate_option iopt;
  iopt.invert = true;
  if (k >= 1) {
    form inv_mz = integrate (zeta*xi, iopt);
    iopt.set_order(2*k+2);
    field lh = integrate (uh*xi, iopt);
    return inv_mz*lh;
  }
  const space& Mh = lambda_h.get_space();
  trial lambda (Mh); test mu (Mh);
  form inv_ms = integrate("sides", lambda*mu, iopt);
  field inv_sh = inv_ms*field(Mh,1);
  field lh = integrate (on_local_sides(inv_sh*lambda_h*xi));
  return (1./(d+1))*lh;
}
```

Example file 2.3: `dirichlet_hdg_average.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "dirichlet_hdg_average.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  field uh, lambda_h;
  din >> catchmark("u")      >> uh
      >> catchmark("lambda") >> lambda_h;
  field bar_uh = dirichlet_hdg_average (uh, lambda_h);
  dout << catchmark("u")      << uh
       << catchmark("bar_u") << bar_uh;
}
```

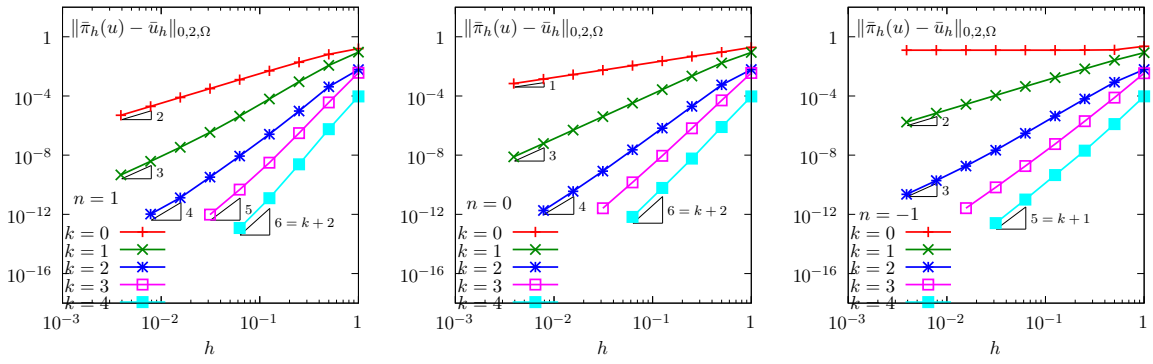


Figure 2.3: Hybrid discontinuous Galerkin method: Poisson problem with Dirichlet boundary conditions in 2D geometry. Super-convergence vs h and k for the piecewise average $\bar{u}_h$ of the approximate solution. (left) $n = 1$; (center) $n = 0$; (right) $n = -1$.

The superconvergence of the piecewise average values of the solution obtained by the hybrid discontinuous Galerkin method was first observed in Cockburn et al. [2008, 2009] and then exploited for many postprocessing application (see e.g. Nguyen et al. [2011]). The file '`dirichlet_hdg_average.icc`' compute $\bar{u}_h = \bar{\pi}_h(u_h, \lambda_h)$, the averaged solution, defined by

(see Cockburn et al. [2008], eqn (2.9b)):

$$\bar{\pi}_h(u_h,\lambda_h) = \begin{cases} \dfrac{1}{d}\displaystyle\sum_{S\subset\partial K} \lambda_h & \text{when } k=0 \\ \dfrac{1}{\text{meas}(K)}\displaystyle\int_K u_h\,\mathrm{d}x & \text{when } k\geqslant 1 \end{cases}$$

The file '`sinusprod_error_hdg_average.cc`' that compute the error $\bar{u}_h - \bar{\pi}_h(u)$ is not listed here but is available in the **Rheolef** example directory. The computation of the error is obtained by:

```
make dirichlet_hdg dirichlet_hdg_average sinusprod_error_hdg_average
mkgeo_grid -t 10 > square.geo
./dirichlet_hdg square.geo P1d | dirichlet_hdg_average | \
    ./sinusprod_error_hdg_average
```

Observe on Fig. 2.3 that, when $n=1$ and for any $k\geqslant 0$, we obtain $\bar{u}_h - \bar{\pi}_h(u)$ of order $k+2$. This is a remarkable result since u_h is piecewise polynomial of order k and is expected to converge at order $k+1$. When $n=0$, for any $k\geqslant 1$ we also observe this superconvergence while, when $k=0$, the average value converges only at first order. Finaly, when $n=-1$, for any $k\geqslant 1$ we do no more observe These observations are consistent with those of [Cockburn et al., 2009, p. 16] (see also [Cockburn et al., 2008, p. 1613]).

2.1.4 Improving the solution by local postprocessing

By combining the approximation $\boldsymbol{\sigma}_h$ of the gradient ∇u, that converges at rate $k+1$ (see Fig. 2.1), with the average $\bar{u}_h$ that super-converges at rate $k+2$ (see Fig. 2.3), it is then possible, by a local integration inside each element, to obtain a new approximation u_h^*, that is piecewise discontinuous polynomial of order $k+1$, and that converges at rate $k+2$.

The postprocess step is nothing than the resolution of the following local Neumann problem in any element $K\in\mathcal{T}_h$ (see [Cockburn et al., 2010, p. 1360], eqn (5.1)):

(P^*): *find* u^*, *defined in* K, *such that*

$$\begin{cases} -\Delta u^* = f \ \text{ in } K \\ \dfrac{\partial u^*}{\partial n} = \boldsymbol{\sigma}_h.\boldsymbol{n} \ \text{ on } \partial K \\ \displaystyle\int_K u^*\,\mathrm{d}x = \int_K \bar{u}_h\,\mathrm{d}x \end{cases}$$

where $\boldsymbol{\sigma}_h$ and $\bar{u}_h$ are given by the previous resolution.

For any $\beta\in\mathbb{R}$, let us introduce the following functional space:

$$X_K^*(\beta) \quad = \quad \left\{ v\in H^1(K)\ ;\ \int_K v\,\mathrm{d}x = \beta \right\}$$

Then, the variational formulation of the problem writes:

(FV^*): *find* $u^*\in X_K\left(\int_K \bar{u}_h\,\mathrm{d}x\right)$ *such that*

$$\int_K \nabla u^*.\nabla v^*\,\mathrm{d}x \quad = \quad \int_K f\,v^*\,\mathrm{d}x + \int_{\partial K} \boldsymbol{\sigma}_h.\boldsymbol{n}\,v^*\,\mathrm{d}s, \quad \forall v^*\in X_K(0)$$

The linear constraint for the imposition of the average value is not easy to impose in $X_K(\beta)$. Indeed, $X_K(\beta)$ is not a vectorial space when $\beta\neq 0$. Following the methodology previously introduced for the Neumann boundary conditions for the Laplace operator, we introduce a Lagrange multiplier denoted here as ζ, which is constant inside each element K.

Finally, the postprocessing step compute the approximation u_h^*, when $\boldsymbol{\sigma}_h$ and u_h are known,as:
$(FV^*)_h$: *find* $(u_h^*, \zeta_h) \in X_h^* \times Z_h$ *such that, on each element* $K \in \mathscr{T}_h$, *we have*

$$\begin{cases} \displaystyle\int_K \nabla u_h^* . \nabla v_h^* \,\mathrm{d}x + \int_K v_h^* \,\zeta_h \,\mathrm{d}x = \int_K f\, v_h^* \,\mathrm{d}x + \int_{\partial K} \boldsymbol{\sigma}_h . \boldsymbol{n}\, v_h^* \,\mathrm{d}s, \quad \forall v_h^* \in X_h^* \\ \displaystyle\int_K u_h^* \,\xi_h \,\mathrm{d}x \qquad\qquad\qquad\qquad = \int_K \bar{u}_h \,\xi_h \,\mathrm{d}x, \quad \forall \xi_h \in X_h^* \end{cases}$$

where

$$\begin{aligned} X_h^* &= \left\{ v \in L^2(\Omega) \;;\; v_{|K} \in P_{k+1},\ \forall K \in \mathscr{T}_h \right\} \\ Z_h &= \left\{ \xi \in L^2(\Omega) \;;\; \xi_{|K} \in P_0,\ \forall K \in \mathscr{T}_h \right\} \end{aligned}$$

Let

$$\begin{aligned} a_h^*(u, \zeta;\, v, \xi) &= \int_\Omega \left(\nabla_h u . \nabla_h v + v\,\zeta + u\,\xi \right) \mathrm{d}x \\ \ell_h^*(v, \xi) &= \int_\Omega \left(f\, v + \bar{u}_h\, \xi \right) \mathrm{d}x + \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\boldsymbol{\sigma}_h . \boldsymbol{n})\, v \,\mathrm{d}s \end{aligned}$$

The second step writes in this abstract setting:
$(FV^*)_h$: *find* $(u_h^*, \zeta_h^*) \in X_h^* \times Z_h$ *such that*

$$a_h^*(u_h^*, \zeta_h;\, v_h^*, \xi_h) = \ell_h^*(v_h^*, \xi_h), \quad \forall (v_h^*, \xi_h) \in X_h^* \times Z_h$$

Note that the matrix associated to the bilinear form a_h^* is symmetric and block-diagonal: it can thus be easily be inverted on the fly at the element level. The present postprocessing stage was first introduced in [Cockburn et al., 2010, p. 1360], eqn (5.1) as a replacement and a simplification of those previously introduced in Cockburn et al. [2008, 2009]. The following code implement this postprocessing.

Example file 2.4: `dirichlet_hdg_post.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "sinusprod_dirichlet.h"
#include "dirichlet_hdg_average.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  Float n, beta;
  field uh, lambda_h, sigma_h;
  din >> catchmark("n")      >> n
      >> catchmark("beta")   >> beta
      >> catchmark("u")      >> uh
      >> catchmark("lambda") >> lambda_h
      >> catchmark("sigma")  >> sigma_h;
  field bar_uh = dirichlet_hdg_average (uh, lambda_h);
  const geo& omega = uh.get_geo();
  size_t d = omega.dimension();
  size_t k = uh.get_space().degree();
  space Xhs (omega, "P"+itos(k+1)+"d"),
        Zhs (omega, "P0"),
        Yhs = Xhs*Zhs;
  trial x(Yhs); test y(Yhs);
  auto us = x[0], zeta = x[1];
  auto vs = y[0], xi   = y[1];
  integrate_option iopt;
  iopt.invert = true;
  form inv_ahs = integrate(dot(grad_h(us),grad_h(vs)) + zeta*vs + xi*us, iopt);
  field lhs = integrate (f(d)*vs + xi*bar_uh
                         + on_local_sides(dot(sigma_h,normal())*vs));
  field xhs = inv_ahs*lhs;
  dout << catchmark("n")      << n << endl
       << catchmark("beta")   << beta << endl
       << catchmark("u")      << xhs[0]
       << catchmark("lambda") << lambda_h
       << catchmark("sigma")  << sigma_h
       << catchmark("zeta")   << xhs[1];
}
```

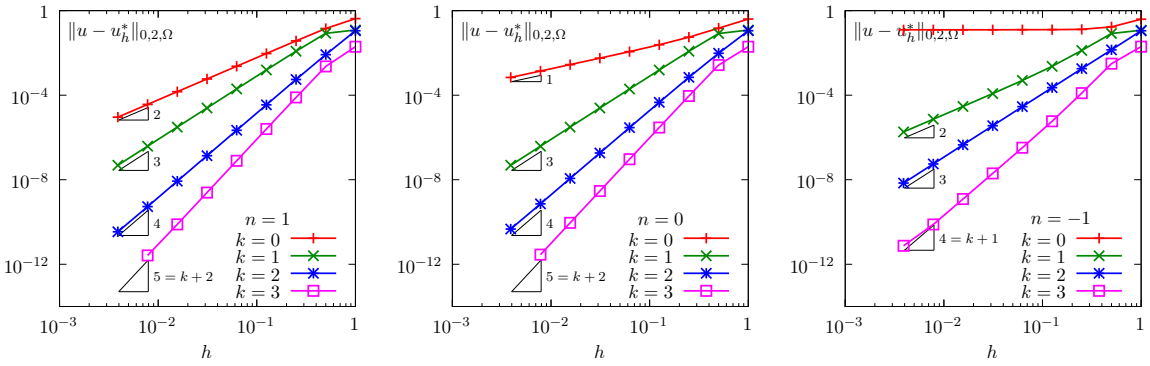


Figure 2.4: Solution post-processing of the hybrid discontinuous Galerkin method: Poisson problem with Dirichlet boundary conditions in 2D geometry. Convergence vs h and k for the post-treated approximate solution. (left) $n = 1$; (center) $n = 0$; (right) $n = -1$.

How to run the program

The compilation and run write:

```
make dirichlet_hdg dirichlet_hdg_post ./sinusprod_error_hdg
mkgeo_grid -t 10 > square.geo
```

```
./dirichlet_hdg square.geo P1d > square.field
./dirichlet_hdg_post < square.field > square-post.field
field square-post.field -elevation
./sinusprod_error_hdg < square.field
./sinusprod_error_hdg < square-post.field
```

The results are shown on Fig. 2.4. When $n = 1$, observe that, for any $k \geqslant 0$, the error for the post-treated solution u_h^* converges to zero at rate $k+2$ in L^2 norm, which is optimal, since u_h^* is a piecewise $k+1$ degree polynomial. When $n = 0$, this result is obtained only for $k \geqslant 1$ while, when $n = -1$, the convergence is suboptimal.

2.1.5 Improving the gradient with the Raviart-Thomas element

Let $(\boldsymbol{\sigma}_h, u_h, \lambda_h)$ be the solution of problem $(FV)_h$. This solution is considered here as known. Observe that the normal component of the numerical flux $\widehat{\boldsymbol{\sigma}}_h.\boldsymbol{n} = \boldsymbol{\sigma}_h.\boldsymbol{n} - \beta h^n(u_h - \lambda_h)$ is continuous across any inter-element boundaries. Indeed, relation (2.3a) is imposed weakely in the variational formulation $(FV)_h$, with a Lagrange multiplier $\mu_h \in M_h$ which is piecewise polynomial of degree k. Since $\widehat{\boldsymbol{\sigma}}_h$ is also piecewise polynomial of degree k, the discrete version of (2.3a) is also satisfied strongly:

$$\widehat{\boldsymbol{\sigma}}_h.\boldsymbol{n} \;=\; [\![\boldsymbol{\sigma}_h - \beta h^n(u_h - \lambda_h)\,\boldsymbol{n}]\!].\boldsymbol{n} \quad = \quad 0 \;\text{ on }\; S,\ \forall S \in \mathscr{S}_h^{(i)}$$

Since any element of $H(\mathrm{div}, \Omega)$ presents the continuity of its normal component across any inter-element boundaries, is possible to define a new $H(\mathrm{div}, \Omega)$ conform approximation of $\boldsymbol{\sigma}$, denoted as $\tilde{\boldsymbol{\sigma}}_h$, which satisfies $\tilde{\boldsymbol{\sigma}}_h.\boldsymbol{n} = \widehat{\boldsymbol{\sigma}}_h.\boldsymbol{n}$ on any internal sides. Similarly to (2.3a), $\tilde{\boldsymbol{\sigma}}_h$ also should satisfy a discrete version of (2.3c)-(2.3c) i.e. $\widehat{\boldsymbol{\sigma}}_h.\boldsymbol{n}$ is equals to g_n on Γ_n and to $\boldsymbol{\sigma}_h.\boldsymbol{n} - \beta h^n(u_h - g_d)$ on Γ_d. It is characterized as the unique element $\tilde{\boldsymbol{\sigma}}_h \in \tilde{T}_h$ satisfying in any element $K \in \mathscr{T}_h$ the following local variational formulation [Cockburn et al., 2010, p. 1360]:

$$\begin{aligned}
\int_K \tilde{\boldsymbol{\sigma}}_h.\tilde{\boldsymbol{\gamma}}_h \,\mathrm{d}x &= \int_K \boldsymbol{\sigma}_h.\tilde{\boldsymbol{\gamma}}_h \,\mathrm{d}x, \quad \forall \tilde{\boldsymbol{\gamma}}_h \in \tilde{G}_h \\
\int_{\partial K} (\tilde{\boldsymbol{\sigma}}_h.\boldsymbol{n})\,\tilde{\mu}_h \,\mathrm{d}s &= \int_{\partial K} (\boldsymbol{\sigma}_h.\boldsymbol{n} - \beta h^n(u_h - \lambda_h))\,\tilde{\mu}_h \,\mathrm{d}s, \quad \forall \tilde{\mu}_h \in \tilde{M}_h
\end{aligned}$$

where we have introduced the finite dimensional spaces

$$\begin{aligned}
\tilde{T}_h &= \left\{ \tilde{\boldsymbol{\tau}}_h \in \left(L^2(\Omega)\right)^d ;\ \tilde{\boldsymbol{\tau}}_{h|K} \in RT_k(K),\ \forall K \in \mathscr{T}_h \right\} \\
\tilde{W}_h &= \left\{ \tilde{\boldsymbol{\gamma}}_h \in \left(L^2(\Omega)\right)^d ;\ \tilde{\boldsymbol{\gamma}}_{h|K} \in P_{k-1}(K),\ \forall K \in \mathscr{T}_h \right\} \\
\tilde{M}_h &= \prod_{K \in \mathscr{T}_h} \left\{ v_{|\partial K} \ ;\ v \in P_k(K) \right\}
\end{aligned}$$

The space $\tilde{T}_h$ contains discontinuous and piecewise k-th order Raviart-Thomas RT_k polynomial vector-valued functions. The space $\tilde{M}_h$ contains, inside each element, the normal trace of functions of $\tilde{T}_h$ while $\tilde{W}_h$ contains piecewise discontinuous polynomial of degree $k-1$. By convention, when $k = 0$, then the space $\tilde{W}_h$ is empty. Observe that $\dim(\tilde{T}_h) = \dim(\tilde{W}_h) + \dim(\tilde{M}_h)$ and that the previous local variational problem is well-posed. Let

$$\begin{aligned}
\tilde{a}_h(\tilde{\boldsymbol{\sigma}}_h;\ [\tilde{\boldsymbol{\gamma}}_h, \tilde{\mu}_h]) &= \int_\Omega \tilde{\boldsymbol{\sigma}}_h.\tilde{\boldsymbol{\gamma}}_h \,\mathrm{d}x + \sum_{K \in \mathscr{T}_h} \sum_{S \subset \partial K} \int_S (\tilde{\boldsymbol{\sigma}}_h.\boldsymbol{n})\,\tilde{\mu}_h \,\mathrm{d}s \\
\tilde{\ell}_h([\tilde{\boldsymbol{\gamma}}_h, \tilde{\mu}_h]) &= \int_\Omega \boldsymbol{\sigma}_h.\tilde{\boldsymbol{\gamma}}_h \,\mathrm{d}x + \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\boldsymbol{\sigma}_h.\boldsymbol{n} - \beta h^n(u_h - \lambda_h))\ \tilde{\mu}_h \,\mathrm{d}s
\end{aligned}$$

For known $(\boldsymbol{\sigma}_h, u_h, \lambda_h)$, the postprocessing of the gradient writes:

$(\widetilde{FV})_h$: *find* $\tilde{\boldsymbol{\sigma}}_h \in \tilde{T}_h$ *such that* $\tilde{a}_h(\tilde{\boldsymbol{\sigma}}_h, [\tilde{\gamma}_h, \tilde{\mu}_h]) = \tilde{\ell}_h([\tilde{\gamma}_h, \tilde{\mu}_h])$ for all $(\tilde{\gamma}_h, \tilde{\mu}_h) \in \tilde{W}_h \times \tilde{M}_h$.
A carreful study shows that the matrix associated with the $\tilde{a}_h$ bilinear form has a block-diagonal structure at the element level and can be efficiently inverted on the fly during the assembly process. This property is due to the usage of discontinuous Raviart-Thomas approximation space $\tilde{T}_h$. Moreover, since the normal component of $\tilde{\boldsymbol{\sigma}}_h$ is equal to the numerical flux $\widehat{\boldsymbol{\sigma}}_h.\boldsymbol{n}$ which is continuous across inter-element boundaries this is also the case for $\tilde{\boldsymbol{\sigma}}_h.\boldsymbol{n}$ and finally $\tilde{\boldsymbol{\sigma}}_h \in H(\mathrm{div}, \Omega)$.

Example file 2.5: `dirichlet_hdg_post_rt.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  Float n, beta;
  field sigma_h, uh, lambda_h;
  din >> catchmark("n")      >> n
      >> catchmark("beta")   >> beta
      >> catchmark("u")      >> uh
      >> catchmark("lambda") >> lambda_h
      >> catchmark("sigma")  >> sigma_h;
  const geo& omega = uh.get_geo();
  size_t d = omega.dimension();
  size_t k = uh.get_space().degree();
  string approx = (k == 0) ? "empty" : "P"+itos(k-1)+"d";
  space Tht(omega, "RT"+itos(k)+"d");
  space Wht(omega, approx, "vector");
  space Mht(omega, "trace(P"+itos(k)+"d)");
  space Sht = Wht*Mht;
  trial sigma_t (Tht); test tau (Sht);
  auto tau_internal = tau[0], tau_n = tau[1];
  auto coef = beta*pow(h_local(),n);
  form  aht = integrate (dot(sigma_t, tau_internal)
                + on_local_sides (dot(sigma_t,normal())*tau_n));
  field lht = integrate(dot(sigma_h, tau_internal)
                + on_local_sides((dot(sigma_h,normal())
                                + coef*(lambda_h - uh))*tau_n));
  field sigma_ht (Tht);
  problem p (aht);
  p.solve (lht, sigma_ht);
  dout << catchmark("n")      << n << endl
       << catchmark("beta")   << beta << endl
       << catchmark("u")      << uh
       << catchmark("lambda") << lambda_h
       << catchmark("sigma")  << sigma_h
       << catchmark("sigmat") << sigma_ht;
}
```

How to run the program

The compilation and run write:

```
make dirichlet_hdg dirichlet_hdg_post_rt
mkgeo_grid -t 10 > square.geo
./dirichlet_hdg square.geo P1d > square.field
./dirichlet_hdg_post_rt < square.field > square-rt.field
field square-rt.field -velocity -mark sigmat -proj P1
make sinusprod_error_hdg_post_rt
./sinusprod_error_hdg_post_rt < square-rt.field
```

Fig. 2.5 plots the errors vs h and k for a two dimensional geometry. Observe that, when $n = 1$, the approximation $\tilde{\boldsymbol{\sigma}}_h$ of the gradient converges to ∇u with a $k+1$ rate in $H($div norm, with:

$$\|\boldsymbol{\tau}\|^2_{\mathrm{div},2,\Omega} = \|\boldsymbol{\tau}\|^2_{0,2,\Omega} + \|\mathrm{div}(\boldsymbol{\tau})\|^2_{0,2,\Omega}$$

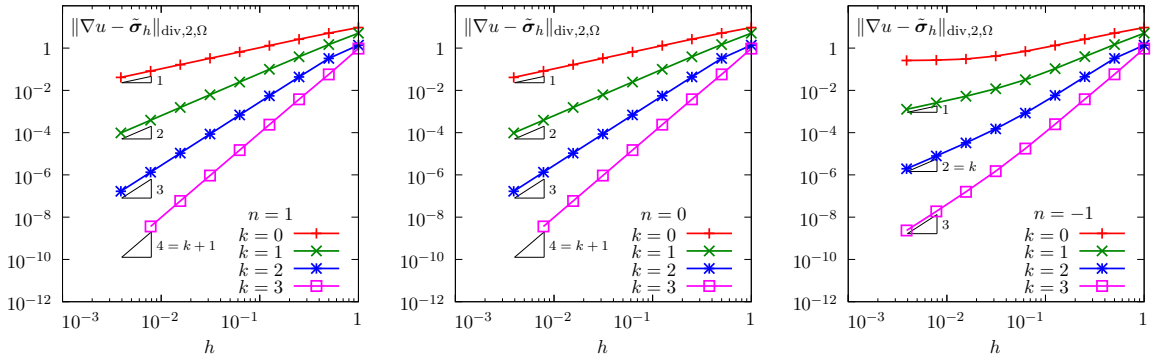


Figure 2.5: Solution post-processing of the hybrid discontinuous Galerkin method: Poisson problem with Dirichlet boundary conditions in 2D geometry. Convergence vs h and k for the post-treated gradient $\tilde{\boldsymbol{\sigma}}_h$ approximate solution. (left) $n = 1$; (center) $n = 0$; (right) $n = -1$.

for any $\boldsymbol{\tau} \in H(\mathrm{div}, \Omega)$. It means that $\mathrm{div}(\tilde{\boldsymbol{\sigma}}_h)$ converges to Δu with a $k+1$ rate in both L^2 norm. This is optimal with respect to the classical interpolation results for the k-th order Raviart-Thomas element (see section 1.1, page 9). When $n = 0$, the convergence is also optimal while, when $n = -1$ it is suboptimal.

n	u_h	$\boldsymbol{\sigma}_h$	u_h^*	$\tilde{\boldsymbol{\sigma}}_h$
1	k	$k+1$	$k+2$	$k+1$
0	$k+1$	$k+1$	$\begin{cases} 1, & k=0 \\ k+2, & k \geqslant 0 \end{cases}$	$k+1$
-1	$\begin{cases} 0, & k=0 \\ k+1, & k \geqslant 0 \end{cases}$	k	$\begin{cases} 0, & k=0 \\ k+1, & k \geqslant 0 \end{cases}$	k

Table 2.1: Hybrid discontinuous Galerkin method: convergence order versus k for $n \in \{1, 0, -1\}$.

Fig. 2.1 summarizes the convergence orders versus the mesh size h in terms of k and n. Observe that $n = 1$ allows one to obtain an optimal convergence of both the the post-processed solution u_h^* and its gradient $\tilde{\boldsymbol{\sigma}}_h$ for any polynomial degree $k \geqslant 0$. When $n = 0$, it is optimal only for $k \geqslant 1$ while when $n = -1$ it is suboptimal. Thus the HDG method and its postprocessed stage could be considered as a whole. It allows, by solving a linear system for the k-th piecewise polynomial approximation of the Lagrange multiplier only, to obtain a k+2 order approximation of the solution in L^2 and a k+1 approximation of its gradient in $H(\mathrm{div})$.

2.2 Hybrid high order (HHO) methods

The aim of this chapter is to introduce to hybrid high order (HHO) methods within the **Rheolef** environment. For a review of hybrid high order methods, and its link to hybrid discontinuous Galerkin (HDG) methods, see Cockburn et al. [2016]. Let us start by some model problems.

2.2.1 The diffusion problem

Let us consider an anisotropic diffusion problem with homogeneous Dirichlet conditions:
(P): *find* u, *defined in* Ω, *such that*

$$\begin{cases} -\mathrm{div}(\boldsymbol{a}\nabla u) = f \ \text{ in } \ \Omega & \text{(2.5a)} \\ u = 0 \ \text{ on } \ \partial\Omega & \text{(2.5b)} \end{cases}$$

where the right-hand-side f and the diffusion $\boldsymbol{a}$ are given functions. The diffusion is assumed to be bounded, symmetric, uniformly positive definite matrix-valued function. Observe that when $\boldsymbol{a} = \boldsymbol{I}$, the problem reduces to the usual Poisson problem with homogeneous Dirichlet conditions.

2.2.2 The reconstruction operator

The cornerstone of the HHO method is the reconstruction operator. Let $k \geqslant 0$ and $\ell \in \{k-1, k, k+1\}$ be two integers and let us introduce the following finite element spaces:

$$\begin{aligned} X_h &= \left\{ v_h \in L^2(\Omega) \ ; \ v_{h|K} \in P_\ell, \ \forall K \in \mathscr{T}_h \right\} \\ M_h &= \left\{ \mu_h \in L^2(\mathscr{S}_h) \ ; \ \mu_{h|S} \in P_k, \ \forall S \in \mathscr{S}_h \right\} \\ X_h^* &= \left\{ v_h^* \in L^2(\Omega) \ ; \ v_{h|K}^* \in P_{k+1}, \ \forall K \in \mathscr{T}_h \right\} \end{aligned}$$

The *reconstruction operator* [Cockburn et al., 2016, p. 637] is defined by

$$\begin{array}{rccc} r_h : & X_h \times M_h & \longrightarrow & X_h^* \\ & (u_h, \lambda_h) & \longmapsto & u_h^* = r_h(u_h, \lambda_h) \end{array}$$

where $u_h^* \in X_h^*$ is characterized by a collection of local constrained minimization problems, associated to local Neumann problems similar to those previously introduced for the HDG method, in subsection 2.1.4, page 20:

$$\begin{aligned} u_h^* &= \inf_{v_h^* \in X_h^*} \frac{1}{2} \int_\Omega |\nabla_h (v_h^* - u_h)|_{\boldsymbol{a}}^2 \, \mathrm{d}x - \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\lambda_h - u_h) \ (\boldsymbol{a} \nabla v_h^*) \cdot \boldsymbol{n} \, \mathrm{d}s \\ &\text{subject to} \int_K v_h^* \, \mathrm{d}x = \int_K u_h \, \mathrm{d}x, \quad \forall K \in \mathscr{T}_h \end{aligned}$$

For any symmetric definite positive matrix $\boldsymbol{c}$, we denote by $|.|_{\boldsymbol{c}}$ the anisotropic norm in $\mathbb{R}^d$ defined by $|\boldsymbol{\xi}|_{\boldsymbol{c}}^2 = \boldsymbol{\xi}.(\boldsymbol{c}\boldsymbol{\xi})$ for all $\boldsymbol{\xi} \in \mathbb{R}^d$. The constraint correspond to the closure for the local Neumann problems, as otherwise the solution would be defined up to a constant. This constrained minimization problem is not convenient for the numerical resolution and we prefer to impose it by introducing a Lagrange multiplier $\zeta_h \in Z_h$ with

$$Z_h = \left\{ \xi_h \in L^2(\Omega) \ ; \ \xi_{h|K} \in P_0, \ \forall K \in \mathscr{T}_h \right\}$$

The corresponding Lagrangian writes

$$\begin{aligned} L_r(u^*, \zeta) &= \frac{1}{2} \int_\Omega |\nabla_h (u^* - u_h)|_{\boldsymbol{a}}^2 \, \mathrm{d}x - \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\lambda_h - u_h) \ (\boldsymbol{a} \nabla u^*) \cdot \boldsymbol{n} \, \mathrm{d}s \\ &\quad + \int_\Omega (u^* - u_h) \, \zeta \, \mathrm{d}x \end{aligned} \tag{2.6}$$

and the reconstruction $u_h^* = r_h(u_h, \lambda_h)$ is characterized as the saddle-point of the Lagrangian:

$$(u_h^*, \zeta_h) \quad = \quad \underset{v_h^* \in X_h^*}{\arg\inf} \sup_{\xi_h Z_h} L_r(v_h^*, \xi_h)$$

Since L_r is differentiable, its saddle point is also characterized as the unique solution of the following collection of local variational problems:

$(FV^*)_h$: *let* $(u_h, \lambda_h) \in X_h \times M_h$ *being given, find* $(u_h^*, \zeta_h) \in X_h^* \times Z_h$ *such that*

$$\begin{cases} \displaystyle\int_K \nabla u_h^* . (\boldsymbol{a} \nabla v_h^*) \, \mathrm{d}x + \int_K v_h^* \zeta_h \, \mathrm{d}x = \int_K \nabla u_h . (\boldsymbol{a} \nabla v_h^*) \, \mathrm{d}x + \int_{\partial K} (\lambda_h - u_h) \ (\boldsymbol{a} \nabla v_h^*) . \boldsymbol{n} \, \mathrm{d}s \\ \displaystyle\int_K u_h^* \xi_h \, \mathrm{d}x \qquad\qquad\qquad\qquad = \int_K u_h \xi_h \, \mathrm{d}x \end{cases}$$

for all $K \in \mathscr{T}_h$ and all $(v_h^*, \xi_h) \in X_h^* \times Z_h$. Let

$$\begin{aligned} a_h^*(u^*, \zeta; \ v^*, \xi) \quad &= \quad \int_\Omega (\nabla_h u^* . (\boldsymbol{a} \nabla_h v^*) + v^* \zeta + u^* \xi) \, \mathrm{d}x \\ b_h^*(u_h, \lambda_h; \ v^*, \xi) \quad &= \quad \int_\Omega (\nabla_h u_h . (\boldsymbol{a} \nabla_h v^*) + u_h \xi) \, \mathrm{d}x + \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\lambda_h - u_h) \ (\boldsymbol{a} \nabla_h v^*) . \boldsymbol{n} \, \mathrm{d}s \end{aligned}$$

Then, the previous reconstruction problem writes:

$(FV^*)_h$: *let* $(u_h, \lambda_h) \in X_h \times M_h$ *being given, find* $(u_h^*, \zeta_h) \in X_h^* \times Z_h$ *such that*

$$a_h^*(u_h^*, \zeta_h; \ v_h^*, \xi_h) \quad = \quad b_h^*(u_h, \lambda_h; \ v_h^*, \xi_h), \quad \forall (v_h^*, \xi_h) \in X_h^* \times Z_h$$

A carreful study shows that the matrix associated to the a_h^* bilinear form has a block-diagonal structure at the element level and its inverse can be easily builded explicitely by inverting on the fly the element matrix during the assembly process. Then, the result of reconstruction operator can be obtained at low computational cost, by simple matrix-vector multiplication.

Example file 2.6: `reconstruction_hho.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "sinusprod.h"
#include "diffusion_isotropic.h"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  string Pkd = (argc > 2) ? argv[2] : "P0",
         Pld = (argc > 3) ? argv[3] :  Pkd;
  space  Xh (omega, Pld),
         Mh (omega["sides"], Pkd);
  size_t k = Xh.degree(), l = Mh.degree(), d = omega.dimension();
  check_macro(l == k-1 || l == k || l == k+1,
    "invalid (k,l) = ("<<k<<","<<l<<")");
  space Xhs(omega, "P"+itos(k+1)+"d"),
        Zh (omega, "P0"),
        Yh = Xhs*Zh;
  trial u(Xh), lambda(Mh), x(Yh);
  test  v(Xh), mu    (Mh), y(Yh);
  auto us = x[0], zeta = x[1];
  auto vs = y[0], xi   = y[1];
  integrate_option iopt;
  iopt.invert = true;
  form inv_m  = integrate (u*v, iopt);
  form ms = integrate (lambda*mu);
  field lh = integrate(u_exact(d)*v);
  field kh = integrate(u_exact(d)*mu);
  field uh = inv_m*lh;
  field lambda_h(Mh);
  problem pms (ms);
  pms.solve (kh, lambda_h);
  form inv_as = integrate (dot(grad_h(us),a(d)*grad_h(vs)) + us*xi
                                + vs*zeta, iopt);
  field lhs   = integrate (dot(grad_h(uh),a(d)*grad_h(vs)) + uh*xi
                     + on_local_sides((lambda_h-uh)
                                      *dot(normal(),a(d)*grad_h(vs))));
  field xh = inv_as*lhs;
  dout << catchmark("us") << xh[0];
}
```

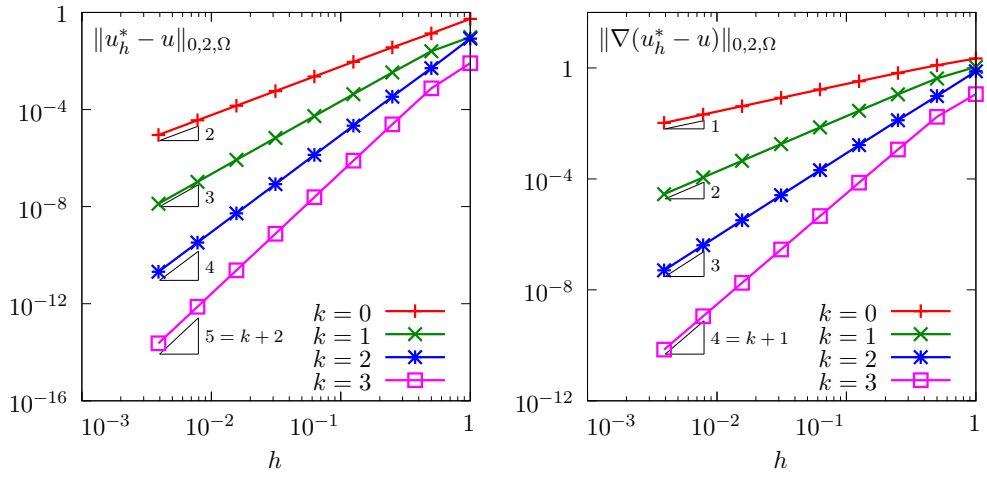


Figure 2.6: Hybrid hight-order (HHO) method: error vs h and k for the reconstruction operator.

How to run the program

The compilation and run write:

```
make reconstruction_hho sinusprod_error_hho_reconstruction
mkgeo_grid -t 10 > square.geo
./reconstruction_hho square.geo P1d P1d | ./sinusprod_error_hho_reconstruction
```

The code '`reconstruction_hho.cc`' first computes u_h and λ_h as L^2 projections on X_h and M_h, respectively, of a given function:

$$u(x) = \prod_{i=0}^{d-1} \sin(\pi x_i)$$

Then, it builds the reconstruction $u_h^* = r_h(u_h, \lambda_h)$ and output the result. Next, the code '`sinusprod_error_hho_reconstruction.cc`' shows the L^2 and H^1 error for the reconstruction u_h^*. The files '`sinusprod.h`', '`diffusion_isotropic.h`', and '`sinusprod_error_hho_reconstruction.cc`', that defines u and $\boldsymbol{a}$, and that compute the errors, respectively, are not listed here but is available in the **Rheolef** example directory. Fig. 2.6 plots the error versus mesh size h and the polynomial order k: the convergence order is $k+2$ in the L^2 norm and $k+1$ in the H^1 one, as expected (see Cockburn et al. [2016], eqn (2.9) or di Pietro and Ern [2015], eqn (3)).

2.2.3 The projections

Let π_{X_h} denotes the L^2 projection from $L^2(\Omega)$ into X_h. For all $u^* \in L^2(\Omega)$, it is defined as $\pi_{X_h}(u^*) = \tilde{u}_h \in X_h$, where $\tilde{u}_h$ is the solution of the following quadratic minimization problem:

$$\tilde{u}_h \quad = \quad \inf_{\tilde{v}_h \in X_h} \frac{1}{2} \int_\Omega (u^* - \tilde{v}_h)^2 \mathrm{d}x$$

Let us consider the special case of computing the projection $\tilde{u}_h = \pi_{X_h}(u_h^*) \in X_h$ of an element $u_h^* \in X_h^*$. The following bilinear forms are introduced:

$$\begin{aligned} m(\tilde{u}, \tilde{v}) &= \int_\Omega \tilde{u}\, \tilde{v}\, \mathrm{d}x \\ c(u^*, \tilde{v}) &= \int_\Omega u^*\, \tilde{v}\, \mathrm{d}x \end{aligned}$$

Then, the projection reduces to:

let $u_h^* \in X_h^*$ being given, find $\tilde{u}_h \in X_h$ such that

$$m(\tilde{u}_h, \tilde{v}_h) \quad = \quad c(u_h^*, \tilde{v}_h), \quad \forall \tilde{v}_h \in X_h$$

Note that the matrix associated to the bilinear form m from $X_h \times X_h$ to $\mathbb{R}$ has a block-diagonal structure at the element level and can be easily inverted on the fly during the assembly process. Then, the result of projection operator can be obtained by simple matrix-vector multiplication.

Let us introduce the space $\tilde{M}_h$ that contains, inside each element, the trace of functions of $\tilde{T}_h$ while $\tilde{W}_h$ contains piecewise discontinuous polynomial of degree k.

$$\tilde{M}_h \quad = \quad \prod_{K \in \mathscr{T}_h} \prod_{S \subset \partial K} P_k(S)$$

Note that this space to those used in the context of the HDG method, in subsection 2.1.5, page 23.

TODO: create a new basis for `space(omega,"broken_trace(Pkd)")` that implements $\tilde{M}_h$.

Let $\pi_{\tilde{M}_h}$ denote the L_2 projection from $L^2(\mathscr{S}_h)$ into $\tilde{M}_h$. We consider the special case of computing the projection $\tilde{\lambda}_h = \pi_{\tilde{M}_h}(u_h^*) \in \tilde{M}_h$ of the restriction to $\mathscr{S}_h$ of an element $u_h^* \in X_h^*$. It is defined as

$$\tilde{\lambda}_h \quad = \quad \arg\inf_{\tilde{\mu}_h \in \tilde{M}_h} \frac{1}{2} \sum_{K \in \mathscr{T}_h} \int_{\partial K} (u_h^* - \tilde{\mu}_h)^2 \mathrm{d}s$$

The following bilinear forms are introduced:

$$\begin{aligned} m_s(\tilde{\lambda}, \tilde{\mu}) &= \sum_{K \in \mathscr{T}_h} \int_{\partial K} \tilde{\lambda}\, \tilde{\mu}\, \mathrm{d}s \\ c_s(u^*, \tilde{\mu}) &= \sum_{K \in \mathscr{T}_h} \int_{\partial K} u^*\, \tilde{\mu}\, \mathrm{d}s \end{aligned}$$

Then, the projection reduces to:

let $u_h^* \in X_h^*$ being given, find $\tilde{\lambda}_h \in \tilde{M}_h$ such that

$$m_s(\tilde{\lambda}_h, \tilde{\mu}_h) \quad = \quad c_s(u_h^*, \tilde{\mu}_h), \quad \forall \tilde{\mu}_h \in \tilde{M}_h$$

Here also, the matrix associated to the bilinear form m_s from $\tilde{M}_h \times \tilde{M}_h$ to $\mathbb{R}$ has a block-diagonal structure at the element level and can be easily inverted on the fly during the assembly process. Then, the result of projection operator can be obtained by simple matrix-vector multiplication.

2.2.4 The discrete problem statement

The discretization of the diffusion problem (2.5a)-(2.5b) by the hybrid high-order (HHO) method [Cockburn et al., 2016, p. 639] is expressed here as a minimization problem:

$$(u_h, \lambda_h) \quad = \quad \arg\min_{(v_h, \mu_h) \in X_h \times \Lambda_h(0)} J_{1,h}(v_h, \mu_h)$$

with

$$\begin{aligned} J_{1,h}(u, \lambda) &= \int_\Omega \left(\frac{1}{2} |\nabla_h r_h(u,\lambda)|_{\boldsymbol{a}}^2 - f\, u \right) \mathrm{d}x + \frac{1}{2} \sum_{K \in \mathscr{T}_h} \sum_{S \subset \partial K} \int_S \beta h_S^{-1} \left(\pi_{\tilde{M}_h} (I - \pi_{X_h})\, r_h(u,\lambda) \right)^2 \mathrm{d}s \\ \Lambda_h(0) &= \left\{ \mu_h \in M_h \ ; \ \mu_{h|S} = 0, \ \forall S \subset \partial\Omega \right\} \end{aligned}$$

The space $\Lambda_h(0)$ is introduced in order to impose the homogeneous Dirichet condition (2.5b). Here $\beta > 0$ is a penalization parameter, which is a constant independent of the mesh size h, while h_K denotes a local mesh characteristic length in the element $K \in \mathscr{T}_h$.

The expression of $J_{1,h}$ is not convenient for practical implementation, as it involves both the reconstruction operator r_h and the projections π_{X_h} and $\pi_{\tilde{M}_h}$. Let us introduce the reconstruction $u_h^* = r_h(u_h, \mu_h)$ as an auxilliary variable and its associated Lagrangian, defined in (2.6), into a new Lagrangian:

$$\begin{aligned} L_{2,h}(u, u^*, \lambda;\ \zeta) &= \int_\Omega \left(\frac{1}{2} |\nabla_h u^*|_{\boldsymbol{a}}^2 - f\, u \right) \mathrm{d}x + \frac{1}{2} \sum_{K \in \mathscr{T}_h} \sum_{S \subset \partial K} \int_S \beta h_S^{-1} \left(\pi_{\tilde{M}_h} (I - \pi_{X_h})\, u^* \right)^2 \mathrm{d}s \\ &\quad + \frac{1}{2} \int_\Omega |\nabla_h (u^* - u)|_{\boldsymbol{a}}^2 \, \mathrm{d}x - \sum_{K \in \mathscr{T}_h} \int_{\partial K} (\lambda - u)\ (\boldsymbol{a} \nabla u^*) \,.\boldsymbol{n}\, \mathrm{d}s \\ &\quad + \int_\Omega (u^* - u)\, \zeta \, \mathrm{d}x \end{aligned}$$

and the previous minimization problem can be equivalently expressed as a saddle point one:

$$(u_h, u_h^*, \lambda_h;\ \zeta_h) \quad = \quad \arg\inf_{\substack{v_h \in X_h \\ v_h^* \in X_h^* \\ \mu_h \in \Lambda_h(0)}} \sup_{\xi_h Z_h} L_{2,h}(v_h, v_h^*, \mu_h;\ \xi_h)$$

The previous expression of $L_{2,h}$ still involves the two projectors π_{X_h} and $\pi_{\tilde{M}_h}$. These explicit call to the projectors are not convenient for the practical implementation. Let us introduce the two additional auxilliary variables $\tilde{u}_h = \pi_{X_h}(u_h^*)$ and $\tilde{\delta}_h = \pi_{\tilde{M}_h}(u_h^* - \tilde{u}_h)$. Then, we obtain the following Lagrangian:

$$\begin{aligned} L_h(u,u^*,\tilde{u},\tilde{\delta},\lambda;\ \zeta) \quad = \quad & \int_\Omega \left(\frac{1}{2}|\nabla_h u^*|^2_{\boldsymbol{a}} - f\,u\right)\,\mathrm{d}x + \frac{1}{2}\sum_{K\in\mathscr{T}_h}\sum_{S\subset\partial K}\int_S \beta h_S^{-1}\,\tilde{\delta}^2\,\mathrm{d}s \\ & + \frac{1}{2}\int_\Omega |\nabla_h(u^*-u)|^2_{\boldsymbol{a}}\,\mathrm{d}x - \sum_{K\in\mathscr{T}_h}\int_{\partial K}(\lambda-u)\ (\boldsymbol{a}\nabla u^*)\,.\boldsymbol{n}\,\mathrm{d}s \\ & + \int_\Omega (u^*-u)\,\zeta\,\mathrm{d}x \\ & + \frac{1}{2}\int_\Omega (\tilde{u}-u^*)^2\mathrm{d}x + \frac{1}{2}\sum_{K\in\mathscr{T}_h}\int_{\partial K}(\tilde{\delta}-u^*+\tilde{u})^2\mathrm{d}s \end{aligned}$$

and the previous minimization problem can be equivalently expressed as a saddle point one:

$$(u_h,u_h^*,\tilde{u}_h,\tilde{\delta}_h,\lambda_h;\ \zeta_h) \quad = \quad \begin{array}{c}\arg\inf \quad \sup \\ {\scriptstyle v_h\,\in\,X_h \quad \xi_h Z_h} \\ {\scriptstyle v_h^*\,\in\,X_h^*} \\ {\scriptstyle \mu_h\,\in\,\Lambda_h(0)} \\ {\scriptstyle \tilde{v}_h\,\in\,X_h} \\ {\scriptstyle \gamma_h\,\in\,\tilde{M}_h}\end{array} L_h(v_h,v_h^*,\tilde{v}_h,\tilde{\gamma}_h,\mu_h;\ \xi_h)$$

Since L_h is differentiable, its saddle point is characterized as the unique solution of the following variational problem:

$(FV)_h$: *find* $(u_h,u_h^*,\tilde{u}_h,\tilde{\delta}_h,\zeta_h) \in X_h\times X_h^*\times X_h\times\tilde{M}_h\times Z_h$ *and* $\lambda_h\in\Lambda_h(0)$ *such that*

$$\begin{cases} a_h([u_h,u_h^*,\tilde{u}_h,\tilde{\delta}_h,\zeta_h];\ [v_h,v_h^*,\tilde{v}_h,\tilde{\gamma}_h,\xi_h]) + b_h([v_h,v_h^*,\tilde{v}_h,\tilde{\gamma}_h,\xi_h];\ \lambda_h) = \ell(v_h,v_h^*,\tilde{v}_h,\tilde{\gamma}_h,\xi_h) \\ b_h([u_h,u_h^*,\tilde{u}_h,\tilde{\delta}_h,\zeta_h];\ \mu_h) = 0 \end{cases}$$

for all $(v_h,v_h^*,\tilde{v}_h,\tilde{\gamma}_h,\xi_h)\in X_h\times X_h^*\times X_h\times\tilde{M}_h\times Z_h$ and $\mu_h\in\Lambda_h(0)$, where

$$\begin{aligned} & a_h([u,u^*,\tilde{u},\tilde{\delta},\zeta];\ [v,v^*,\tilde{v},\tilde{\gamma},\xi]) \\ = & \int_\Omega \Big(\nabla_h u^*.(\boldsymbol{a}\nabla_h v^*) + \nabla_h(u^*-u).(\boldsymbol{a}\nabla_h(v^*-v)) + (u^*-u)\,\xi + (v^*-v)\,\zeta \\ & \qquad + \ (\tilde{u}-u^*)(\tilde{v}-v^*)\Big)\,\mathrm{d}x \\ & + \sum_{K\in\mathscr{T}_h}\sum_{S\subset\partial K}\int_S \Big(\beta h_S^{-1}\,\tilde{\delta}\,\tilde{\gamma} + u\ (\boldsymbol{a}\nabla v^*)\,.\boldsymbol{n} + v\ (\boldsymbol{a}\nabla u^*)\,.\boldsymbol{n} \ + \ (\tilde{\delta}-u^*+\tilde{u})(\tilde{\gamma}-v^*+\tilde{v})\Big)\,\mathrm{d}s \end{aligned}$$

$$b_h([u,u^*,\tilde{u},\tilde{\delta},\zeta];\ \mu) = -\sum_{K\in\mathscr{T}_h}\int_{\partial K}\mu\ (\boldsymbol{a}\nabla u^*)\,.\boldsymbol{n}\,\mathrm{d}s$$

$$\ell_h(v,v^*,\tilde{v},\tilde{\gamma},\xi) = \int_\Omega f\,v\,\mathrm{d}x$$

A carreful study shows that the matrix A_h associated to the a_h bilinear form is symmetric indefinite and has a block-diagonal structure at the element level. Its inverse A_h^{-1} can be easily builded explicitely by inverting on the fly the element matrix during the assembly process. Let us denote by B_h the matrix associated to the bilinear form b_h and by L_h the vector associated to the right-hand side. Let us denote $\chi_h = (u_h,u_h^*,\tilde{u}_h,\tilde{\delta}_h,\zeta_h)$ for convenience. The previous problem writes in matrix-vector form:

$$\begin{aligned} & \begin{pmatrix} A_h & B_h^T \\ B_h & 0 \end{pmatrix}\begin{pmatrix} \chi_h \\ \lambda_h \end{pmatrix} = \begin{pmatrix} \ell_h \\ 0 \end{pmatrix} \\ \iff \quad & \begin{cases} (B_hA_h^{-1}B_h^T)\lambda_h = B_hA_h^{-1}\ell_h \\ \chi_h = A_h^{-1}(\ell_h - B_h^T\lambda_h) \end{cases} \end{aligned}$$

The only remaining unknown is the Lagrange multiplier λ_h in a linear system involving the Schur complement matrix $S_h = B_h A_h^{-1} B_h^T$. Then, the groups of all variables contained in χ_h is simply obtained by a direct computation. The following code implements this technics:

Example file 2.7: `dirichlet_hho.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "dirichlet_homogeneous.h"
#include "diffusion_isotropic.h"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  string Pkd = (argc > 2) ? argv[2] : "P1d",
         Pld = (argc > 3) ? argv[3] :  Pkd;
  Float beta = (argc > 4) ? atof(argv[4]) : 1;
  space  Xh (omega, Pld),
         Mh (omega["sides"], Pkd);
  Mh.block("boundary");
  size_t k = Xh.degree(), l = Mh.degree(), d = omega.dimension();
  check_macro(l == k-1 || l == k || l == k+1,
    "invalid (k,l) = ("<<k<<","<<l<<")");
  space Xhs(omega, "P"+itos(k+1)+"d"),
        Zh (omega, "P0"),
        Mht(omega, "trace(P"+itos(k)+"d)");
  space Yh = Xh*Xhs*Xh*Mht*Zh;
  trial x(Yh), lambda(Mh);
  test  y(Yh), mu(Mh);
  auto u = x[0], us = x[1], ut = x[2], deltat = x[3], zeta = x[4];
  auto v = y[0], vs = y[1], vt = y[2], gammat = y[3], xi    = y[4];
  integrate_option iopt;
  iopt.invert = true;
  form inv_a = integrate(dot(grad_h(us),a(d)*grad_h(vs))
                + dot(grad_h(us)-grad_h(u),a(d)*(grad_h(vs)-grad_h(v)))
                + (us-u)*xi + (vs-v)*zeta + (ut-us)*(vt-vs)
                + on_local_sides(beta/h_local()*deltat*gammat
                  + u*dot(a(d)*grad_h(vs),normal())
                  + v*dot(a(d)*grad_h(us),normal())
                  + (deltat - us + ut)*(gammat - vs + vt)), iopt);
  form b = integrate(omega,on_local_sides(-mu*dot(a(d)*grad_h(us),normal())));
        // TODO: remove omega, autodetect it ?
  field lh = integrate (f(d)*v);
        // TODO: f -> -f ? check the FV...
  field lambda_h(Mh,0);
  form s = b*inv_a*trans(b);
  field rh = b*(inv_a*lh);
  problem p (s);
  p.solve (rh, lambda_h);
  field xh = inv_a*(lh - b.trans_mult(lambda_h));
  dout << catchmark("beta")   << beta << endl
       << catchmark("u")      << xh[0]
       << catchmark("us")     << xh[1]
       << catchmark("ut")     << xh[2]
       << catchmark("delta")  << xh[3]
       << catchmark("zeta")   << xh[4]
       << catchmark("lambda") << lambda_h;
}
```

Example file 2.8: `dirichlet_homogeneous.h`

```
struct f {
  Float operator() (const point& x) const { return 1; }
  f(size_t=0) {}
};
struct g {
  Float operator() (const point& x) const { return 0; }
  g(size_t=0) {}
};
```

As usual, the right-hand-side f and the Dirichlet boundary condition g has been chosen such that the exact solution is given by:

$$u(x) = \prod_{i=0}^{d-1} \sin(\pi x_i)$$

The files '`sinusprod_dirichlet.h`', that defines the data f and g, and '`sinusprod_error_hdg.cc`', that compute the error in various norms, are not listed here but are available in the **Rheolef** example directory.

How to run the program

The compilation and run write:

```
make dirichlet_hho
mkgeo_grid -t 10 > square.geo
./dirichlet_hho square.geo 1 1 > square.field
field square.field -elevation
field square.field -elevation -mark lambda
```

Bibliography

B. Cockburn, B. Dong, and J. Guzmán. A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. *Math. Comput.*, 77(264):1887–1916, 2008.

B. Cockburn, J. Guzmán, and H. Wang. Superconvergent discontinuous Galerkin methods for second-order elliptic problems. *Math. Comput.*, 78(265):1–24, 2009.

B. Cockburn, J. Gopalakrishnan, and F.-J. Sayas. A projection-based error analysis of HDG methods. *Math. Comput.*, 79(271):1351–1367, 2010.

B. Cockburn, D. A. di Pietro, and A. Ern. Bridging the hybrid high-order and hybridizable discontinuous Galerkin methods. *ESAIM Math. Model. Numer. Anal.*, 50(3):635–650, 2016.

D. A. di Pietro and A. Ern. Hybrid high-order methods for variable-diffusion problems on general meshes. *C. R. Math.*, 353(1):31–34, 2015.

N. C. Nguyen, J. Peraire, and B. Cockburn. Hybridizable discontinuous Galerkin methods. In *Spectral and high order methods for partial differential equations*, pages 63–84. Springer, 2011.

P.-A. Raviart and J.-M. Thomas. A mixed finite element method for 2-nd order elliptic problems. In *Mathematical aspects of finite element methods*, pages 292–315. Springer, 1977.

J. E. Roberts and J.-M. Thomas. Mixed and hybrid methods. In P. G. Ciarlet and J.-L. Lions, editors, *Handbook of numerical analysis. Volume 2. Finite element methods (part 1)*, chapter 4, pages 524–639. Elsevier, 1991.

B. Stroustrup. C++ programming styles and libraries. *InformIt.com*, 0:0, 2002.

N. Wirth. *Algorithm + data structure = programs*. Prentice Hall, NJ, USA, 1985.

List of example files

Index