HAL
open science

Efficient C++ finite element computing with Rheolef: volume 2: discontinuous Galerkin methods

Pierre Saramito

To cite this version:

Pierre Saramito. Efficient C++ finite element computing with Rheolef: volume 2: discontinuous Galerkin methods. DEA. Grenoble, France, France. 2015, pp.56. cel-00863021v3

HAL Id: cel-00863021

https://cel.hal.science/cel-00863021v3

Submitted on 3 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

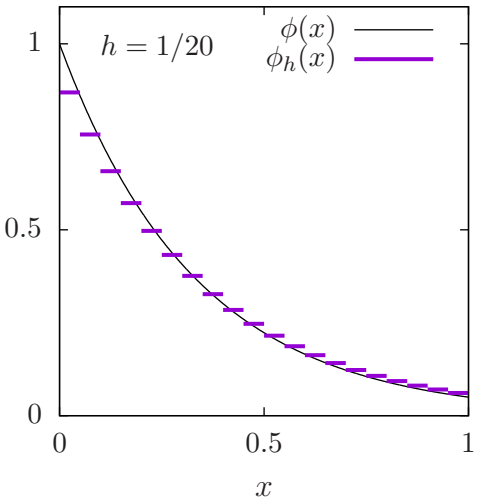
Efficient C++ finite element computing with Rheolef

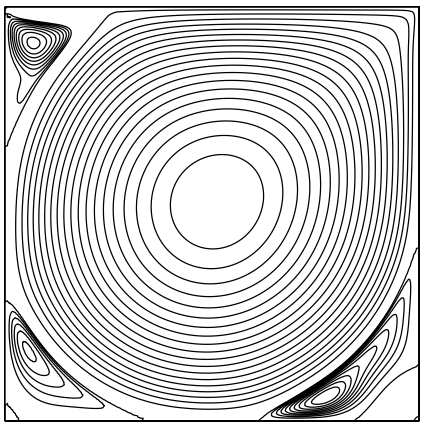
volume 2:

discontinuous Galerkin methods

Pierre Saramito

version 6.7 update 24 March 2016





$Re = 10\,000$

Copyright (c) 2003-2013 Pierre Saramito

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

Notations **2**

I Getting started with simple problems **5**

1 Scalar first-order problems **7**

1.1 The transport equation . . . 7

1.2 Nonlinear scalar hyperbolic problems . . . 10

1.3 Example: the Burgers equation . . . 14

2 Scalar second-order problems **21**

2.1 The Poisson problem with Dirichlet boundary conditions . . . 21

2.2 The Helmholtz problem with Neumann boundary conditions . . . 23

2.3 Nonlinear scalar hyperbolic problems with diffusion . . . 25

2.4 Example: the Burgers equation with diffusion . . . 25

II Fluids and solids computations **31**

3 The linear elasticity and the Stokes problems **33**

3.1 The linear elasticity problem . . . 33

3.2 The Stokes problem . . . 35

3.3 The stationnary Navier-Stokes problem . . . 37

III Technical appendices **53**

A GNU Free Documentation License **55**

List of example files **62**

List of commands **64**

Index **64**

Notations

Rheolef	mathematics	description
`d`	$d \in \{1,2,3\}$	dimension of the physical space
`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 in $\Omega \subset \mathbb{R}^d$
`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 $d=3$
`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

Rheolef	mathematics	description
`grad_s(u)`	$\nabla_s \mathbf{u} = \nabla \mathbf{u} P$	tangential gradient of a vector
`Ds(u)`	$D_s(\mathbf{u}) = P D(\mathbf{u}) P$	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)`	$[\![\phi]\!] = \phi_{\vert K_0} - \phi_{\vert K_1}$	jump accros inter-element side $S = \partial K_0 \cap K_1$
`average(phi)`	$\{\!\{\phi\}\!\} = (\phi_{\vert K_0} + \phi_{\vert K_1})/2$	average accross S
`inner(phi)`	$\phi_{\vert K_0}$	inner trace on S
`outer(phi)`	$\phi_{\vert K_1}$	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)`	$(\nabla_h \phi)_{\vert K} = \nabla(\phi_{\vert K}), \forall K \in \mathcal{T}_h$	broken gradient
`div_h(u)`	$(\mathrm{div}_h \mathbf{u})_{\vert K} = \mathrm{div}(\mathbf{u}_{\vert K}), \forall K \in \mathcal{T}_h$	broken divergence of a vector field
`Dh(u)`	$(D_h(\mathbf{u}))_{\vert K} = D(\mathbf{u}_{\vert K}), \forall K \in \mathcal{T}_h$	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)`	$\sinh(\phi)$	
`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)`	$\min(\phi, \psi)$	

Rheolef	mathematics	description
`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

Part I

Getting started with simple problems

Chapter 1

Scalar first-order problems

The aim of this chapter is to introduce to discontinuous Galerkin methods within the **Rheolef** environment. For some recent presentations of discontinuous Galerkin methods, see [11] for theoretical aspects and [16] for algorithmic and implementation. The discontinuous Galerkin methods are in active development in **Rheolef**, and new features will appear soon.

1.1 The transport equation

The steady scalar transport problem writes:
(P): *find ϕ, defined in Ω, such that*

$$\begin{aligned} \mathbf{u}.\nabla\phi + \sigma\phi &= f \quad \text{in } \Omega \\ \phi &= \phi_\Gamma \quad \text{on } \partial\Omega_- \end{aligned}$$

where $\mathbf{u}$, $\sigma > 0$, f and ϕ_Γ being known. Notice that this is the steady version of the unsteady diffusion-convection problem previously introduced in section 6.2, page 86 and when the diffusion coefficient ν vanishes. Here, the $\partial\Omega_-$ notation is the *upstream* boundary part, defined by

$$\partial\Omega_- = \{x \in \partial\Omega;\ \mathbf{u}(x).\mathbf{n}(x) < 0\}$$

Let us suppose that $\mathbf{u} \in W^{1,\infty}(\Omega)^d$ and introduce the space:

$$X = \{\varphi \in L^2(\Omega);\ (\mathbf{u}.\nabla)\varphi \in L^2(\Omega)^d\}$$

and, for all $\phi, \varphi \in X$

$$\begin{aligned} a(\phi,\varphi) &= \int_\Omega (\mathbf{u}.\nabla\phi\,\varphi + \sigma\,\phi\,\varphi)\ \mathrm{d}x &+ \int_{\partial\Omega} \max(0, -\mathbf{u}.\mathbf{n})\,\phi\ \varphi\,\mathrm{d}s \\ l(\varphi) &= \int_\Omega f\,\varphi\,\mathrm{d}x &+ \int_{\partial\Omega} \max(0, -\mathbf{u}.\mathbf{n})\,\phi_\Gamma\,\varphi\,\mathrm{d}s \end{aligned}$$

Then, the variational formulation writes:
(FV): *find $\phi \in X$ such that*

$$a(\phi,\varphi) = l(\varphi),\ \forall\varphi \in X$$

Notice that the term $\max(0, -\mathbf{u}.\mathbf{n}) = (|\mathbf{u}.\mathbf{n}| - \mathbf{u}.\mathbf{n})/2$ is positive and vanishes everywhere except on $\partial\Omega_-$. Thus, the boundary condition $\phi = \phi_\Gamma$ is weakly imposed on $\partial\Omega_-$ via the integrals on the boundary. The *discontinuous* finite element space is defined by:

$$X_h = \{\varphi_h \in L^2(\Omega); \varphi_{h|K} \in P_k,\ \forall K \in \mathcal{T}_h\}$$

where $k \geqslant 0$ is the polynomial degree. Notice that $X_h \not\subset X$ and that the $\nabla\phi_h$ term has no more sense for discontinous functions $\phi_h \in X_h$. Following [11, p. 14], we introduce the *broken gradient* ∇_h as a convenient notation:

$$(\nabla_h \phi_h)_{|K} = \nabla(\phi_{h|K}),\ \forall K \in \mathcal{T}_h$$

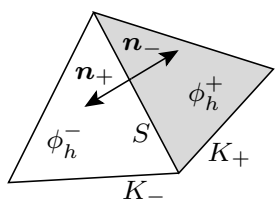
Thus

$$\int_\Omega \mathbf{u}.\nabla_h \phi_h\, \varphi_h \,\mathrm{d}x = \sum_{K\in\mathcal{T}_h} \int_K \mathbf{u}.\nabla\phi_h\, \varphi_h \,\mathrm{d}x,\ \forall \phi_h, \varphi_h \in X_h$$

This leads to a discrete version a_h of the bilinear form a, defined for all $\phi_h, \varphi_h \in X_h$ by (see e.g. [11, p. 57], eqn. (2.34)):

$$\begin{aligned} a_h(\phi_h, \varphi_h) &= \int_\Omega (\mathbf{u}.\nabla_h\phi_h\, \varphi_h + \sigma\phi_h\ \varphi_h)\ \mathrm{d}x + \int_{\partial\Omega} \max(0, -\mathbf{u}.\mathbf{n})\, \phi_h\, \varphi_h \,\mathrm{d}s \\ &+ \sum_{S\in\mathscr{S}_h^{(i)}} \int_S \Big(-\ \mathbf{u}.\mathbf{n}\, [\![\phi_h]\!]\, \{\!\{\varphi_h\}\!\} + \frac{\alpha}{2}|\mathbf{u}.\mathbf{n}|\, [\![\phi_h]\!]\, [\![\varphi_h]\!] \Big)\ \mathrm{d}s \end{aligned}$$

The last term involves a sum over $\mathscr{S}_h^{(i)}$, the set of *internal sides* of the mesh $\mathcal{T}_h$. Each internal side



$\boldsymbol{n} = \boldsymbol{n}_- = -\boldsymbol{n}_+$ on S

Figure 1.1: Discontinuous Galerkin method: an internal side, its two neighbor elements and their opposite normals.

$S \in \mathscr{S}_h^{(i)}$ has two possible orientations: one is choosen definitively. In practice, this orientation is defined in the '`.geo`' file containing the mesh, where all sides are listed, together with their orientation. Let $\mathbf{n}$ the normal to the oriented side S: as S is an internal side, there exists two elements K_- and K_+ such that $S = \partial K_- \cap \partial K_+$ and $\mathbf{n}$ is the outward unit normal of K_- on $\partial K_- \cap S$ and the inward unit normal of K_+ on $\partial K_+ \cap S$, as shown on Fig. 1.1. For all $\phi_h \in X_h$, recall that ϕ_h is in general discontinuous accross the internal side S. We define on S the *inner value* $\phi_h^- = \phi_{h|K_-}$ of ϕ_h as the restriction $\phi_{h|K_-}$ of ϕ_h in K_- along $\partial K_- \cap S$. Conversely, we define the *outer value* $\phi_h^+ = \phi_{h|K_+}$. We also denote on S the *jump* $[\![\phi_h]\!] = \phi_h^- - \phi_h^+$ and the *average* $\{\!\{\phi_h\}\!\} = (\phi_h^- + \phi_h^+)/2$. The last term in the definition of a_h is ponderated by a coefficient $\alpha \geqslant 0$. Choosing $\alpha = 0$ correspond to the so-called *centered flux* approximation, while $\alpha = 1$ is the *upwinding flux* approximation. The case $\alpha = 1$ and $k = 0$ (piecewise constant approximation) leads to the popular upwinding finite volume scheme. Finally, the discrete variational formulation writes:

$(FV)_h$: *find* $\phi_h \in X_h$ *such that*

$$a_h(\phi_h, \varphi_h) = l(\varphi_h),\ \forall \varphi_h \in X_h$$

The following code implement this problem in the **Rheolef** environment.

Example file 1.1: `transport_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  Float alpha   = (argc > 3) ? atof(argv[3]) : 1;
  Float sigma   = (argc > 4) ? atof(argv[4]) : 3;
  point u (1,0,0);
  trial phi (Xh); test psi (Xh);
  form ah = integrate (dot(u,grad_h(phi))*psi + sigma*phi*psi)
          + integrate ("boundary", max(0, -dot(u,normal()))*phi*psi)
          + integrate ("internal_sides",
                - dot(u,normal())*jump(phi)*average(psi)
                + 0.5*alpha*abs(dot(u,normal()))*jump(phi)*jump(psi));
  field lh = integrate ("boundary", max(0, -dot(u,normal()))*psi);
  solver sah (ah.uu());
  field phi_h(Xh);
  phi_h.set_u() = sah.solve(lh.u());
  dout << catchmark("sigma") << sigma << endl
       << catchmark("phi")   << phi_h;
}
```

Comments

The data are $\phi_\gamma = 1$ and $\mathbf{u} = (1, 0, 0)$, and then the exact solution is known: $\phi(x) = \exp(-\sigma x_0)$. The numerical tests are running with $\sigma = 3$ by default. The one-dimensional case writes:

```
make transport_dg
mkgeo_grid -e 10 > line.geo
./transport_dg line P0  | field -
./transport_dg line P1d | field -
./transport_dg line P2d | field -
```

Observe the jumps accross elements: these jumps decreases with mesh refinement or when polynomial approximation increases. The two-dimensional case writes:

```
mkgeo_grid -t 10 > square.geo
./transport_dg square P0  | field -elevation -
./transport_dg square P1d | field -elevation -
./transport_dg square P2d | field -elevation -
```

The elevation view shows details on inter-element jumps. Finaly, the three-dimensional case writes:

```
mkgeo_grid -T 5 > cube.geo
./transport_dg cube P0  | field -
./transport_dg cube P1d | field -
./transport_dg cube P2d | field -
```

Fig. 1.2 plots the solution when $d = 1$ and $k = 0$: observe that the boundary condition $\phi = 1$ at $x_0 = 0$ is only weakly satified. It means that the approximation $\phi_h(0)$ tends to 1 when h tnds to zero. Fig. 1.3 plots the error $\phi - \phi_h$ in L^2 and L^∞ norms: these errors behave as $\mathcal{O}\left(h^{k+1}\right)$ for all $k \geqslant 0$, which is optimal. A theoretical $\mathcal{O}\left(h^{k+1/2}\right)$ error bound was shown in [17]. The present numerical results confirm that these theoretical error bounds can be improved for some families of meshes, as pointed out by Richter [19], that showed a $\mathcal{O}\left(h^{k+1}\right)$ optimal bound for the transport problem. This result was recently extended by Cockburn *et al.* [6], while Peterson [18] showed that the estimate $\mathcal{O}\left(h^{k+1/2}\right)$ is sharp for general families of quasi-uniform meshes.

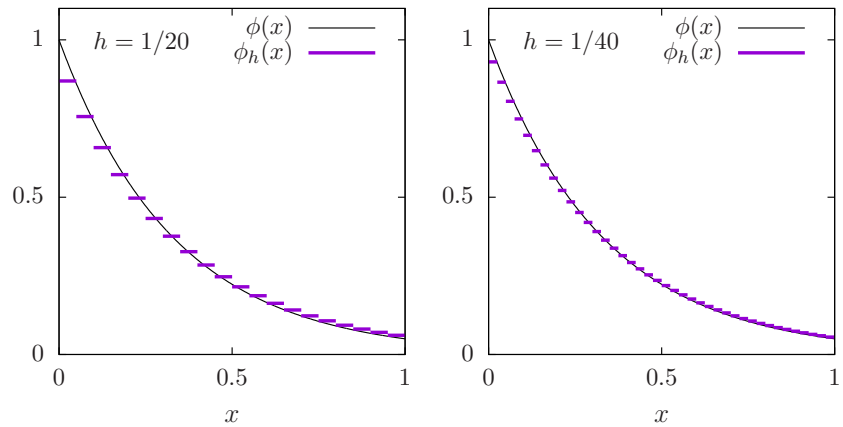


Figure 1.2: The discontinuous Galerkin method for the transport problem when $k = 0$ and $d = 1$.

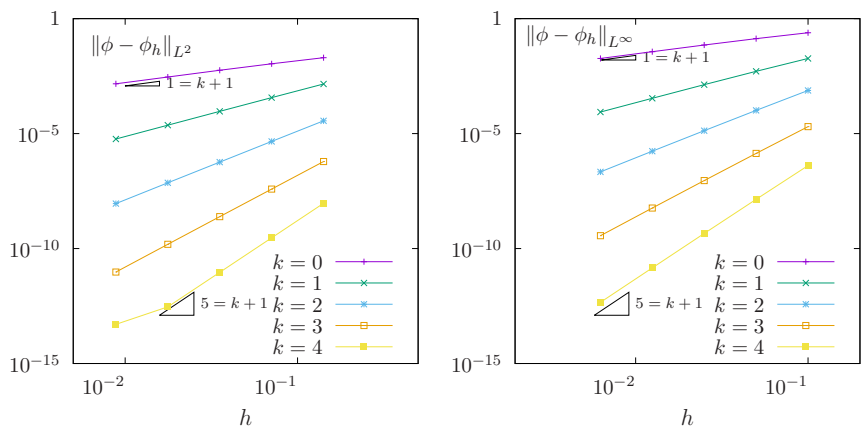


Figure 1.3: The discontinuous Galerkin method for the transport problem: convergence when $d = 2$.

1.2 Nonlinear scalar hyperbolic problems

The aim of this paragraph is to study the discontinuous Galerkin discretization of scalar nonlinear hyperbolic equations. This section presents the general framework and discretization tools while the next section illustrates the method for the Burgers equation.

1.2.1 Problem setting

A time-dependent nonlinear hyperbolic problem writes in general form [11, p. 99]:

(P): *find u, defined in $]0,T[\times\Omega$, such that*

$$\frac{\partial u}{\partial t} + \operatorname{div} \mathbf{f}(u) = 0 \quad \text{in }]0,T[\times\Omega \tag{1.1a}$$

$$u(t{=}0) = u_0 \quad \text{in } \Omega \tag{1.1b}$$

$$\mathbf{f}(u).\mathbf{n} = \Phi(\mathbf{n};\, u, g) \text{ on }]0,T[\times\partial\Omega \tag{1.1c}$$

where $T > 0$, $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$ and the initial condition $\mathbf{u}_0$ being known. As usual, $\mathbf{n}$ denotes the outward unit normal on the boundary $\partial\Omega$. The function $\mathbf{f} : \mathbb{R} \longrightarrow \mathbb{R}^d$ is also known and supposed

to be continuously differentiable. The initial data u_0, defined in Ω, and the boundary one, g, defined on $\partial\Omega$ are given. The function Φ, called the Godunov flux associated to $\mathbf{f}$, is defined, for all $\boldsymbol{\nu} \in \mathbb{R}^d$ and $a, b \in \mathbb{R}$, by

$$\Phi(\boldsymbol{\nu};\, a, b) = \begin{cases} \min\limits_{v\in[a,b]} \mathbf{f}(v).\boldsymbol{\nu} & \text{when } a \leqslant b \\ \max\limits_{v\in[b,a]} \mathbf{f}(v).\boldsymbol{\nu} & \text{otherwise} \end{cases} \tag{1.1d}$$

1.2.2 Space discretization

In this section, we consider first the semi-discretization with respect to space while the problem remains continuous with respect to time. The semi-discrete problem writes in variational form [11, p. 100]:

$(P)_h$: *find* $u_h \in C^1([0,T], X_h)$ *such that*

$$\begin{aligned} \int_\Omega \frac{\partial u_h}{\partial t} v_h \,\mathrm{d}x - \int_\Omega \mathbf{f}(u_h).\nabla_h v_h \,\mathrm{d}x + \sum_{S\in\mathcal{S}_h^{(i)}} \Phi(\mathbf{n};\, u_h^-, u_h^+)[\![v_h]\!] \,\mathrm{d}s + \int_{\partial\Omega} \Phi(\mathbf{n};\, u_h, g) v_h \,\mathrm{d}s &= 0, \quad \forall v_h \in X_h \\ u_h(t{=}0) &= \pi_h(u_0) \end{aligned}$$

where π_h denotes the Lagrange interpolation operator on X_h and others notations has been introduced in the previous section.

For convenience, we introduce the discrete operator G_h, defined for all $u_h, v_h \in X_h$ by

$$\int_\Omega G_h(u_h) v_h \,\mathrm{d}x = -\int_\Omega \mathbf{f}(u_h).\nabla_h v_h \,\mathrm{d}x + \sum_{S\in\mathcal{S}_h^{(i)}} \int_S \Phi(\mathbf{n};\, u_h^-, u_h^+)[\![v_h]\!] \,\mathrm{d}s + \int_{\partial\Omega} \Phi(\mathbf{n};\, u_h, g) v_h \,\mathrm{d}s \tag{1.2}$$

For a given $u_h \in X_h$, we also define the linear form g_h as

$$g(v_h) = \int_\Omega G_h(u_h) v_h \,\mathrm{d}x$$

As the matrix M, representing the L^2 scalar product in X_h, is block-diagonal, it can be easily inverted at the element level, and for a given $u_h \in X_h$, we have $G(u_h) = M^{-1} g_h$. Then, the problem writes equivalently as a set of coupled nonlinear ordinary differential equations.

$(P)_h$: *find* $u_h \in C^1([0,T], X_h)$ *such that*

$$\frac{\partial u_h}{\partial t} + G_h(u_h) = 0$$

1.2.3 Time discretization

Let $\Delta t > 0$ be the time step. The previous nonlinear ordinary differential equations are discretized by using a specific explicit Runge-Kutta with intermediates states [13,14,22]. This specific variant of the usual Runge-Kutta scheme, called *strong stability preserving*, is suitable for avoiding possible spurious oscillations of the approximate solution when the exact solution has a poor regularity. Let u_h^n denotes the approximation of u_h at time $t_n = n\Delta t$, $n \geqslant 0$. Then u_h^{n+1} is defined by recurrence:

$$\begin{aligned} u_h^{n,0} &= u_h^n \\ u_h^{n,i} &= \sum_{j=0}^{i-1} \alpha_{i,j} u_h^{n,j} - \Delta t\, \beta_{i,j} G_h\left(u_h^{n,j}\right), \quad 1 \leqslant i \leqslant p \\ u_h^{n+1} &= u_h^{n,p} \end{aligned}$$

where the coefficients satisfy $\alpha_{i,j} \geqslant 0$ and $\beta_{i,j} \geqslant 0$ for all $1 \leqslant i \leqslant p$ and $0 \leqslant j \leqslant i-1$, and $\sum_{j=0}^{i-1} \alpha_{i,j} = 1$ for all $1 \leqslant i \leqslant p$. Notice that when $p = 1$ this scheme coincides with the usual explicit Euler scheme. For a general $p \geqslant 1$ order scheme, there are $p-1$ intermediate states $u_h^{n,i}$, $i = 1 \ldots p-1$. Computation of the coefficients $\alpha_{i,j}$ and $\beta_{i,j}$ can be founded in [13,14,22] and are grouped in file '`runge_kutta_ssp.icc`' of the examples directory for convenience.

1.2.4 Slope limiters

Slope limiters are required when the solution develops discontinuities: this is a very classical feature of most solutions of nonlinear hyperbolic equations. A preliminary version of the slope limiter proposed by Cocburn et al. [5, p. 208] is implemented in **Rheolef**:Âăthis preliminary version only supports the $d = 1$ dimension and $k = 1$ polynomial degree. Recall that the $k = 0$ case do not need any limiter. More general implementation will support the $d = 2, 3$ and $k \geqslant 2$ cases in the future. The details of the limiter implementation is presented in this section: the impatient reader, who is interested by applications, can jump to the next section, devoted to the Burgers equation.

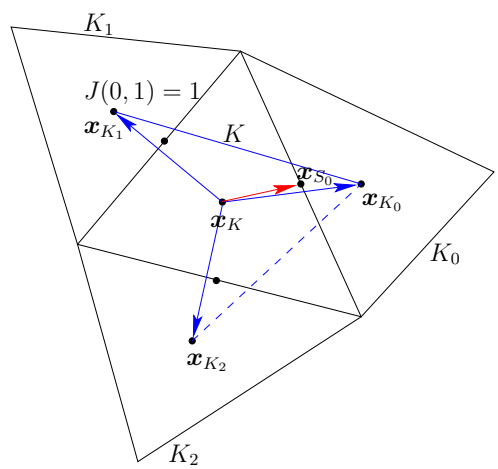


Figure 1.4: Limiter: the neighbors elements and the middle edge points.

Fig. 1.4 shows the $d+1$ neighbor elements K_i, $i = 0 \ldots d$ around an element d. Let $S_i = \partial K \cap \partial K_i$, $i = 0 \ldots d$ be the i-th side of K. We denote by $\boldsymbol{x}_K$, $\boldsymbol{x}_{K_i}$ ad $\boldsymbol{x}_{S_i}$ the barycenters of these elements and sides, $i = 0 \ldots d$. When $d = 2$, the barycenter $\boldsymbol{x}_{S_i}$ of the edge belongs to the interior of a triangle $(\boldsymbol{x}_K, \boldsymbol{x}_{K_i}, \boldsymbol{x}_{K_{J_{i,1}}})$ for exactly one of the two possible $J_{i,1} \neq i$ and $0 \leqslant J_{i,1} \leqslant d$. When $d = 3$, the barycenter $\boldsymbol{x}_{S_i}$ of the face belongs to the interior of a tetrahedron $(\boldsymbol{x}_K, \boldsymbol{x}_{K_i}, \boldsymbol{x}_{K_{J_{i,1}}}, \boldsymbol{x}_{K_{J_{i,2}}})$ for exactly one pair $(J_{i,1}, J_{i,2})$, up to a permutation, of the three possible pairs $J_{i,1}, J_{i,2} \neq i$ and $0 \leqslant J_{i,1}, J_{i,2} \leqslant d$. Let us denote $J_{i,0} = i$. Then, the vector $\overrightarrow{\boldsymbol{x}_K \boldsymbol{x}_{S_i}}$ decompose on the basis $(\overrightarrow{\boldsymbol{x}_K \boldsymbol{x}_{K_{J_{i,k}}}})_{0 \leqslant k \leqslant d-1}$ as

$$\overrightarrow{\boldsymbol{x}_K \boldsymbol{x}_{S_i}} = \sum_{k=0}^{d-1} \alpha_{i,k} \overrightarrow{\boldsymbol{x}_K \boldsymbol{x}_{K_{J_{i,k}}}} \tag{1.3}$$

where $\alpha_{i,k} \geqslant 0$, $k = 0 \ldots d-1$. Let us consider now the patch ω_K composed of K and its d neighbors:

$$\omega_K = K \cup K_0 \cup \ldots \cup K_d$$

For any affine function $\xi \in P_1(\omega_K)$ over this patch, let us denote

$$\begin{aligned}\delta_{K,i}(\xi) &= \sum_{k=0}^{d-1} \alpha_{i,k} \left(\xi(\boldsymbol{x}_{K_{J_{i,k}}}) - \xi(\boldsymbol{x}_K) \right), \quad i = 0 \ldots d-1 \\ &= \xi(\boldsymbol{x}_{S_i}) - \xi(\boldsymbol{x}_K) \quad \text{from (1.3)}\end{aligned}$$

In other terms, $\delta_{K,i}(\xi)$ represents the departure of the value of ξ at $\boldsymbol{x}_{S_i}$ from its average $\xi(\boldsymbol{x}_K$ on the element K.

Let now $(\varphi_i)_{0 \leqslant i \leqslant d-1}$ denote the Lagrangian basis in K associated to the set of nodes $(\boldsymbol{x}_{S_i})_{0 \leqslant i \leqslant d-1}$:

$$\begin{aligned}\varphi_i(\boldsymbol{x}_{S_j}) = \delta_{i,j}, \quad 0 \leqslant i, j \leqslant d-1 \\ \sum_{i=0}^{d-1} \varphi_i(\boldsymbol{x}) = 1, \quad \forall \boldsymbol{x} \in K\end{aligned}$$

The affine function $\xi \in P_1(\omega_K)$ expresses on this basis as

$$\xi(\boldsymbol{x}) = \xi(\boldsymbol{x}_K) + \sum_{i=0}^{d-1} \delta_{K,i}(\xi)\, \varphi_i(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in K$$

Let now $u_h \in \mathbb{P}_{1d}(\mathscr{T}_h)$. On any element $K \in \mathscr{T}_h$, let us introduce its average value:

$$\bar{u}_K = \frac{1}{\text{meas(K)}} \int_K u_h(\boldsymbol{x})\, \mathrm{d}x$$

and its departure from its average value:

$$\tilde{u}_K(\boldsymbol{x}) = u_{h|K}(\boldsymbol{x}) - \bar{u}_K, \quad \forall \boldsymbol{x} \in K$$

Notice that $u_h \notin P_1(\omega_K)$. Let us extends $\delta_{K,i}$ to u_h as

$$\delta_{K,i}(u_h) = \sum_{k=0}^{d-1} \alpha_{i,k} \left(\bar{u}_{K_{J_{i,k}}} - \bar{u}_K \right), \quad i = 0 \ldots d-1$$

Since $u_h \notin P_1(\omega_K)$, we have $\tilde{u}_K(\boldsymbol{x}_{K_{J_{i,k}}}) \neq \delta_{K,i}(u_h)$ in general. The idea is then to capture oscillations by controlling the departure of the values $\tilde{u}_K(\boldsymbol{x}_{K_{J_{i,k}}})$ from the values $\delta_{K,i}(u_h)$. Thus, associate to $u_h \in \mathbb{P}_{1d}(\mathscr{T}_h)$ the quantities

$$\Delta_{K,i}(u_h) = \text{minmod}_{\text{TVB}} \left(\tilde{u}_K(\boldsymbol{x}_{K_{J_{i,k}}}),\ \theta \delta_{K,i}(u_h) \right)$$

for all $i = 0 \ldots d-1$ and where $\theta \geqslant 1$ is a parameter of the limiter and

$$\text{minmod}_{\text{TVB}}(a, b) = \begin{cases} a & \text{when } |a| \leqslant M h^2 \\ \text{minmod}(a, b) & \text{otherwise} \end{cases}$$

where $M > 0$ is a tunable parameter which can be evaluated from the curvature of the initial datum at its extrema by setting

$$M = \sup_{\boldsymbol{x} \in \Omega, \nabla u_0(\boldsymbol{x}) = 0} |\nabla \otimes \nabla u_0| \tag{1.4}$$

Introduced in [21], the basic idea is to deactivate the limiter when space derivatives are of order h^2. This improves the limiter behavior near smooth local extrema. The minmod function is defined by

$$\text{minmod}(a, b) = \begin{cases} \text{sgn}(a) \min(|a|, |b|) & \text{when } \text{sgn}(a) = \text{sgn}(b) \\ 0 & \text{otherwise} \end{cases}$$

Then, for all $i = 0 \dots d-1$ we define

$$\begin{aligned} r_K(u_h) &= \frac{\displaystyle\sum_{j=0}^{d-1} \max(0, -\Delta_{K,j}(u_h))}{\displaystyle\sum_{j=0}^{d-1} \max(0, \Delta_{K,j}(u_h))} \geqslant 0 \\ \hat{\Delta}_{K,i}(u_h) &= \min(1, r_K(u_h)) \max(0, \Delta_{K,i}(u_h)) \\ &\quad - \min(1, 1/r_K(u_h)) \max(0, -\Delta_{K,i}(u_h)), \quad i = 0 \dots d-1 \text{ when } r_K(u_h) \neq 0 \end{aligned}$$

Finally, the limited function $\Lambda_h(u_h)$ is defined element by element for all element $K \in \mathscr{T}_h$ for all $\boldsymbol{x} \in K$ by

$$\Lambda_h(u_h)_{|K}(\boldsymbol{x}) = \begin{cases} \bar{u}_K + \displaystyle\sum_{i=1}^{d-1} \Delta_{K,i}(u_h)\ \varphi_i(\boldsymbol{x}) & \text{when } r_K(u_h) = 0 \\ \bar{u}_K + \displaystyle\sum_{i=1}^{d-1} \hat{\Delta}_{K,i}(u_h)\ \varphi_i(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

Notice that there are two types of computations involved in the limiter: one part is independent of u_h and depends only upon the mesh: $J_{i,k}$ and $\alpha_{i,k}$ on each element. It can be computed one time for all. The other part depends upon the values of u_h.

Notice that the limiter preserves the average value of u_h on each element K and also the functions that are globally affine on the patch ω_K. Also we have, inside each element K and for all side index $i = 0 \dots d-1$:

$$\left|\Lambda_h(u_h)_{|K}(\boldsymbol{x}_{S_i}) - \bar{u}_K\right| \leqslant \max\left(|\Delta_{K,i}(u_h)|, |\hat{\Delta}_{K,i}(u_h)|\right) \leqslant |\Delta_{K,i}(u_h)| \leqslant \left|u_{h|K}(\boldsymbol{x}_{S_i}) - \bar{u}_K\right|$$

It means that, inside each element, the gradient of the P_1 limited function is no larger than that of the original one.

The limiter on an element close to the boundary should takes into account the inflow condition. In [7], the modifications are described.

1.3 Example: the Burgers equation

As an illustration, let us consider now the test with the one-dimensional ($d = 1$) Burgers equation for a propagating slant step (see e.g. [3, p. 87]) in $\Omega =]0, 1[$. We have $\mathbf{f}(u) = u^2/2$, for all $u \in \mathbb{R}$. In that case, the Godunov flux (1.1d), introduced page 11, can be computed explicitly for any $\boldsymbol{\nu} = (\nu_0) \in \mathbb{R}^d$ and $a, b \in \mathbb{R}$:

$$\Phi(\boldsymbol{\nu}; a, b) = \begin{cases} \nu_0 \min\left(a^2, b^2\right)/2 & \text{when } \nu_0 \geqslant 0 \text{ and } a \leqslant b \text{ or } \nu_0 \leqslant 0 \text{ and } a \geqslant b \\ \nu_0 \max\left(a^2, b^2\right)/2 & \text{otherwise} \end{cases}$$

Example file 1.2: `burgers.icc`

```
point f (const Float& u) { return point (sqr(u)/2); }
```

Example file 1.3: `burgers_flux_godunov.icc`

```
Float phi (const point& nu, Float a, Float b) {
  if ((nu[0] >= 0 && a <= b) || (nu[0] <= 0 && a >= b))
    return nu[0]*min(sqr(a),sqr(b))/2;
  else
    return nu[0]*max(sqr(a),sqr(b))/2;
}
```

1.3.1 Computing an exact solution

An exact solution is useful for testing numerical methods. The computation of such an exact solution for the one dimensional Burgers equation is described by Hartens *et al.* [15]. The authors consider first the problem with a periodic boundary condition:

(P): find $u:\ ]0,T[\times]-1,1[\longmapsto \mathbb{R}$ such that

$$\begin{aligned}
\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) &= 0 \ \text{ in }]0,T[\times]-1,1[\\
u(t{=}0,x) &= \alpha + \beta\sin(\pi x + \gamma), \quad \text{a.e. } x \in]-1,1[\\
u(t,x{=}-1) &= u(t,x{=}1) \ \text{ a.e. } t \in]0,T[
\end{aligned}$$

where α, β and γ are real parameters. Let us denote w the solution of the problem when $\beta = 1$ and $\alpha = \gamma = 0$; i.e. with the initial condition $w(t{=}0,x) = \sin(\pi x)$, a.e. $x \in]-1,1[$. For any $x \in [0,1[$ and $t > 0$, the solution $\bar{w} = w(t,x)$ satisfies the characteristic relation

$$\bar{w} = \sin(\pi(x - \bar{w}t))$$

This nonlinear relation can be solved by a Newton algorithm. Then, for $x \in]-1,0[$, the solution is completed by symmetry: $w(t,x) = -w(t,-x)$. Finally, the general solution for any α, β and $\gamma = 0$ writes $u(t,x) = \alpha + w(\beta t,\ x - \alpha t + \gamma)$. File '`harten.icc`' implements this approach.

Example file 1.4: `harten.icc`

```
#include "harten0.icc"
struct harten {
  Float operator() (const point& x) const {
    Float x0 = x[0]-a*t+c;
    Float shift = -2*floor((x0+1)/2);
    Float xs = x0 + shift;
    check_macro (xs >= -1 && xs <= 1, "invalid xs="<<xs);
    return a + b*h0 (point(xs));
  }
  harten (Float t1=0, Float a1=1, Float b1=0.5, Float c1=0):
    h0(b1*t1), t(t1), a(a1), b(b1), c(c1) {}
  Float M() const { constexpr Float pi = acos(-1.0); return sqr(pi)*b; }
  Float min() const { return a-b; }
  Float max() const { return a+b; }
protected:
  harten0 h0;
  Float t, a, b, c;
};
using u_init = harten;
using g = harten;
```

Example file 1.5: `harten_show.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "harten.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  size_t nmax = (argc > 3) ? atoi(argv[3]) : 1000;
  Float tf    = (argc > 4) ? atof(argv[4]) : 2.5;
  Float a     = (argc > 5) ? atof(argv[5]) : 1;
  Float b     = (argc > 6) ? atof(argv[6]) : 0.5;
  Float c     = (argc > 7) ? atof(argv[7]) : 0;
  branch even("t","u");
  for (size_t n = 0; n <= nmax; ++n) {
    Float t = n*tf/nmax;
    field pi_h_u = interpolate (Xh, harten(t,a,b,c));
    dout << even(t,pi_h_u);
  }
}
```

The included file '`harten0.icc`' is not shown here by is available in the example directory.

Comments

Notice that the constant M, used by the limiter in (1.4), can be explicitly computed for this solution: $M = \beta\pi^2$.
The animation of this exact solution is performed by the following commands:

```
make harten_show
mkgeo_grid -e 2000 -a -1 -b 1 > line2.geo
./harten_show line2 P1 1000 2.5 > line2-exact.branch
branch line2-exact -gnuplot
```

Fig. 1.5 shows the solution u for $\alpha = 1, \beta = 1/2$ and $\gamma = 0$. It is regular until $t = 2/\pi$ (Fig. 1.5.c) and then develops a chock for $t > 2/\pi$ (Fig. 1.5.d). After its apparition, this chock interacts with the expansion wave in $]-1, 1[$: this brings about a fast decay of the solution (Figs. 1.5.e and f). Fig. 1.5 plots also a numerical solution: its computation is the aim of the next section.

1.3.2 Numerical resolution

When replacing the periodic boundary condition with a inflow one, associated with the boundary data g, we choose g to be the value of the exact solution of the problem with periodic boundary conditions: $g(t, x) = \alpha + w(\beta t,\, x - \alpha t)$ for $x \in \{-1, 1\}$.

Example file 1.6: `burgers_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "harten.icc"
#include "burgers.icc"
#include "burgers_flux_godunov.icc"
#include "runge_kutta_ssp.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  Float cfl = 1;
  limiter_option_type lopt;
  size_t nmax = (argc > 3) ? atoi(argv[3]) : numeric_limits<size_t>::max();
  Float  tf   = (argc > 4) ? atof(argv[4]) : 2.5;
  size_t p    = (argc > 5) ? atoi(argv[5]) : pmax;
  lopt.M      = (argc > 6) ? atoi(argv[6]) : u_init().M();
  if (nmax == numeric_limits<size_t>::max()) {
      nmax = floor(1+tf/(cfl*omega.hmin()));
  }
  Float delta_t = tf/nmax;
  form_option_type fopt;
  fopt.invert = true;
  trial u (Xh); test v (Xh);
  form inv_m = integrate (u*v, fopt);
  vector<field> uh(p+1, field(Xh,0));
  uh[0] = interpolate (Xh, u_init());
  branch even("t","u");
  dout << catchmark("delta_t") << delta_t << endl
       << even(0,uh[0]);
  for (size_t n = 1; n <= nmax; ++n) {
    for (size_t i = 1; i <= p; ++i) {
      uh[i] = 0;
      for (size_t j = 0; j < i; ++j) {
        field lh =
         - integrate (dot(compose(f,uh[j]),grad_h(v)))
         + integrate ("internal_sides",
             compose (phi, normal(), inner(uh[j]), outer(uh[j]))*jump(v))
         + integrate ("boundary",
             compose (phi, normal(), uh[j], g(n*delta_t))*v);
        uh[i] += alpha[p][i][j]*uh[j] - delta_t*beta[p][i][j]*(inv_m*lh);
      }
      uh[i] = limiter(uh[i], g(n*delta_t)(point(-1)), lopt);
    }
    uh[0] = uh[p];
    dout << even(n*delta_t,uh[0]);
  }
}
```

Comments

The Runge-Kutta time discretization combined with the discontinuous Galerkin space discretization is implemented for this test case.

The P_0 approximation is performed by the following commands:

```
make burgers_dg
mkgeo_grid -e 200 -a -1 -b 1 > line2-200.geo
./burgers_dg line2-200.geo P0 1000 2.5 > line2-200-P0.branch
branch line2-200-P0.branch -gnuplot
```

The two last commands compute the P_0 approximation of the solution, as shown on Fig. 1.5. Observe the robust behavior of the solution at the vicinity of the chock. By replacing `P0` by `P1d` in the previous commands, we obtain the P_1-discontinuous approximation.

```
./burgers_dg line2-200.geo P1d 1000 2.5 > line2-200-P1d.branch
branch line2-200-P1d.branch -gnuplot
```

Fig. 1.6 plots the error vs h for $k = 0$ and $k = 1$. Fig. 1.6.a plots in a time interval $[0, T]$ with $T = 1/\pi$, before the chock that occurs at $t = 2/\pi$. In that interval, the solution is regular and the error approximation behaves as $\mathcal{O}(h^{k+1})$. The time interval has been chosen sufficiently small for the error to depend only upon h. Fig. 1.6.b plots in a larger time interval $[0, T]$ with $T = 5/2$, that includes the chock. Observe that the error behaves as $\mathcal{O}(h)$ for both $k = 0$ and 1. This is optimal when $k = 0$ but not when $k = 1$. This is due to the loss of regularity of the exact solution that presents a discontinuity; A similar observation can be found in [26], table 4.1.

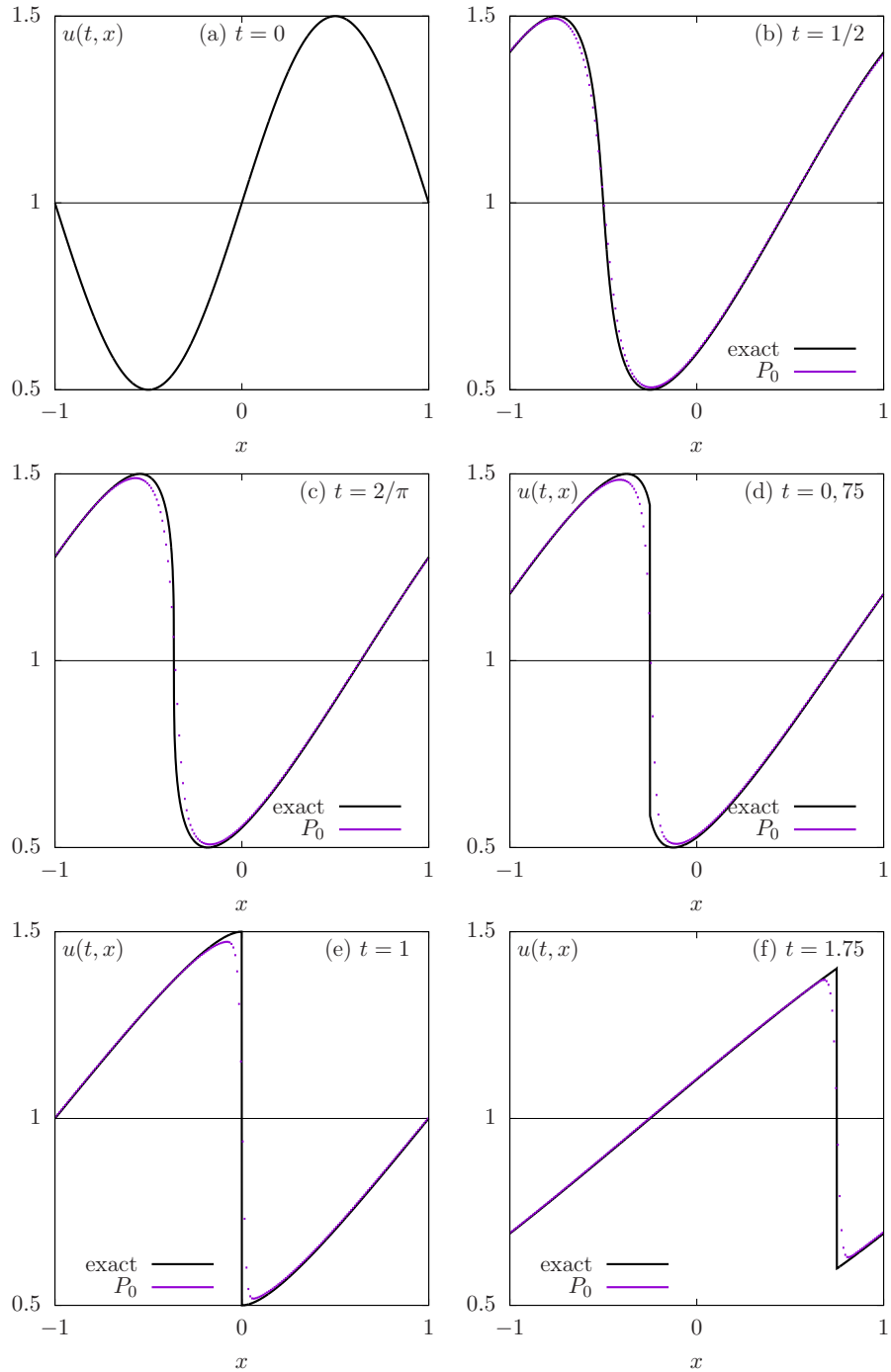


Figure 1.5: Harten's exact solution of the Burgers equation ($\alpha = 1, \beta = 1/2, \gamma = 0$). Comparison with the P_0 appromation ($h = 1/100$, RK-SSP(3)).

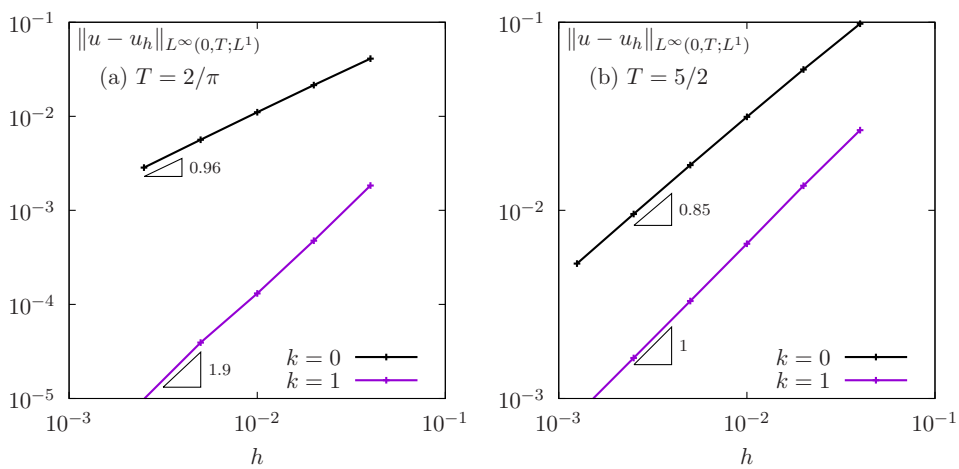


Figure 1.6: Burgers equation: error between the P_0 approximation and the exact solution of the Harten's problem ($\alpha = 1, \beta = 1/2, \gamma = 0$): (a) before chock, with $T = 1/pi$; (b) after chock, with $T = 5/2$.

Chapter 2

Scalar second-order problems

2.1 The Poisson problem with Dirichlet boundary conditions

The Poisson problem with non-homogeneous Dirichlet boundary conditions has been already introduced in volume 1, section 2.1, page 27:

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

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega \end{aligned}$$

where f and g are given.

The *discontinuous* finite element space is defined by:

$$X_h = \{v_h \in L^2(\Omega); v_{h|K} \in P_k,\ \forall K \in \mathcal{T}_h\}$$

where $k \geqslant 1$ is the polynomial degree. As elements of X_h do not belongs to $H^1(\Omega)$, due to discontinuities at inter-elements, we introduce the broken Sobolev space:

$$H^1(\mathcal{T}_h) = \{v \in L^2(\Omega);\ v_{|K} \in H^1(K),\ \forall K \in \mathcal{T}_h\}$$

such that $X_h \subset H^1(\mathcal{T}_h)$. We introduce the folowing bilinear form $a_h(.,.)$ and linear for $l_h(.)$, defined for all $u, v \in H^1(\mathcal{T}_h)$ by (see e.g. [11, p. 125 and 127], eqn. (4.12)):

$$a_h(u,v) = \int_\Omega \nabla_h u.\nabla_h v \,\mathrm{d}x + \sum_{S\in\mathscr{S}_h} \int_S \left(\eta_s \,[\![u]\!]\,[\![v]\!] - \{\!\{\nabla_h u.\mathbf{n}\}\!\}\,[\![v]\!] - [\![u]\!]\,\{\!\{\nabla_h v.\mathbf{n}\}\!\}\right) \mathrm{d}s \quad (2.1)$$

$$l_h(v) = \int_\Omega f\, u \,\mathrm{d}x + \int_{\partial\Omega} \left(\eta_s\, g\, v - g\, \nabla_h v.\mathbf{n}\right) \mathrm{d}s \quad (2.2)$$

The last term involves a sum over $\mathscr{S}_h$, the set of *all sides* of the mesh $\mathcal{T}_h$, i.e. the internal sides and the boundary sides. By convenience, the definition of the jump and average are extended to all boundary sides as $[\![u]\!] = \{\!\{u\}\!\} = u$. Notice that, as for the previous transport problem, the Dirichlet boundary condition $u = g$ is weakly imposed on $\partial\Omega$ via the integrals on the boundary. Finally, $\eta_s > 0$ is a stabilization parameter on a side S. The stabilization term associated to η_s is present in order to achieve coercivity: it penalize interface and boundary jumps. A common choice is $\eta_s = \beta\, h_s^{-1}$ where $\beta > 0$ is a constant and h_s is a local length scale associated to the current side S. One drawnback to this choice is that it requires the end user to specify the numerical constant β. From one hand, if the value of this parameter is not sufficiently large, the form $a_h(.,.)$ is not coercive and the approximate solution develops instabilities an do not converge [12]. From other hand, if the value of this parameter is too large, its affect the overall efficiency of the iterative solver of the linear system: the spectral condition number of the matrix associated to $a_h(.,.)$

grows linearly with this paramater [4]. An explicit choice of penalty parameter is proposed in [20]: $\eta_s = \beta\,\varpi_s$ where $\beta = (k+1)(k+d)/d$ and

$$\varpi_s \quad = \quad \begin{cases} \dfrac{\text{meas}(\partial K)}{\text{meas}(K)} & \text{when } S = K \cap \partial\Omega \text{ is a boundary side} \\ \max\left(\dfrac{\text{meas}(\partial K_0)}{\text{meas}(K_0)}, \dfrac{\text{meas}(\partial K_1)}{\text{meas}(K_1)}\right) & \text{when } S = K_0 \cap K_1 \text{ is an internal side} \end{cases}$$

Notice that ϖ_s scales as h_s^{-1}. Now, the computation of the penalty parameter is fully automatic and the convergence of the method is always guaranted to converge. Moreover, this choice has been founded to be sharp and it preserves the optimal efficiency of the iterative solvers. Finally, the discrete variational formulation writes:

$(FV)_h$: *find* $u_h \in X_h$ *such that*

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

The following code implement this problem in the **Rheolef** environment.

Example file 2.1: `dirichlet_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "cosinusprod_laplace.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  size_t d = omega.dimension();
  size_t k = Xh.degree();
  Float beta = (k+1)*(k+d)/d;
  trial u (Xh); test v (Xh);
  form  a = integrate (dot(grad_h(u),grad_h(v)))
          + integrate ("sides", beta*penalty()*jump(u)*jump(v)
                - jump(u)*average(dot(grad_h(v),normal()))
                - jump(v)*average(dot(grad_h(u),normal())));
  field lh = integrate (f(d)*v)
           + integrate ("boundary", beta*penalty()*g(d)*v
                - g(d)*dot(grad_h(v),normal()));
  solver sa (a.uu());
  field uh(Xh);
  uh.set_u() = sa.solve(lh.u());
  dout << uh;
}
```

Comments

The `penalty()` pseudo-function implements the computation of ϖ_s in **Rheolef**. The right-hand side f and g are given by (2.1), volume 1, page 28. In that case, the exact solution is known. Running the one-dimensional case writes:

```
make dirichlet_dg
mkgeo_grid -e 10 > line.geo
./dirichlet_dg line P1d | field -
./dirichlet_dg line P2d | field -
```

Fig. 2.1 plots the one-dimensional solution when $k = 1$ for two meshes. Observe that the jumps at inter-element nodes decreases very fast with mesh refinement and are no more perceptible on the plots. Recall that the Dirichlet boundary conditions at $x = 0$ and $x = 1$ is only weakly imposed: the corresponding jump at the boundary is also not perceptible.

The two-dimensional case writes:

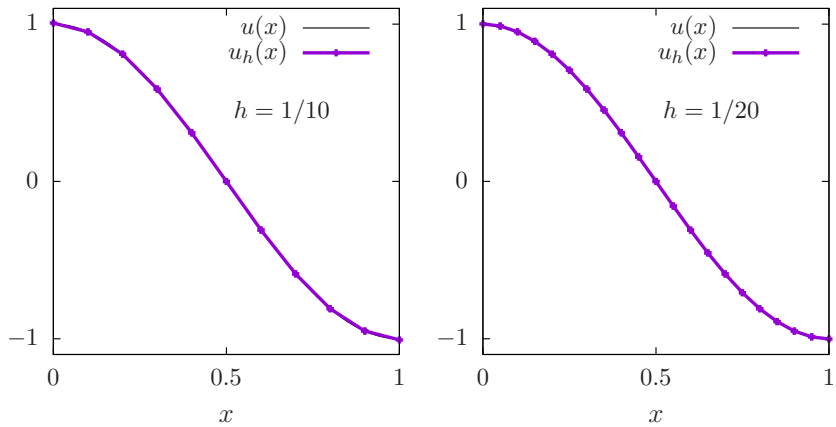


Figure 2.1: The discontinuous Galerkin method for the Poisson problem when $k = 1$ and $d = 1$.

```
mkgeo_grid -t 10 > square.geo
./dirichlet_dg square P1d | field -elevation -
./dirichlet_dg square P2d | field -elevation -
```

and the three-dimensional one

```
mkgeo_grid -T 10 > cube.geo
./dirichlet_dg cube P1d | field -
./dirichlet_dg cube P2d | field -
```

Error analysis

The space $H^1(\mathcal{T}_h)$ is equiped with the norm $\|.\|_{1,h}$, defined for all $v \in H^1(\mathcal{T}_h$ by [11, p. 128]:

$$\|v\|_{1,h}^2 = \|\nabla_h v\|_{0,\Omega}^2 + \sum_{S \in \mathscr{S}_h} \int_S h_s^{-1} \, [\![v]\!]^2 \, \mathrm{d}s$$

The code `cosinusprod_error_dg.cc` compute the error in these norms. This code it is not listed here but is available in the **Rheolef** example directory. The computation of the error writes:

```
make cosinusprod_error_dg
./dirichlet_dg square P1d | cosinusprod_error_dg
```

Fig. 2.2 plots the error $u - u_h$ in L^2, L^∞ and the $\|.\|_{1,h}$ norms. The L^2 and L^∞ error norms behave as $\mathcal{O}\left(h^{k+1}\right)$ for all $k \geqslant 0$, while the $\|.\|_{1,h}$ one behaves as $\mathcal{O}\left(h^k\right)$, which is optimal.

2.2 The Helmholtz problem with Neumann boundary conditions

The Poisson problem with non-homogeneous Neumann boundary conditions has been already introduced in volume 1, section 2.2, page 35:

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

$$\begin{aligned} u - \Delta u &= f \text{ in } \Omega \\ \frac{\partial u}{\partial n} &= g \text{ on } \partial\Omega \end{aligned}$$

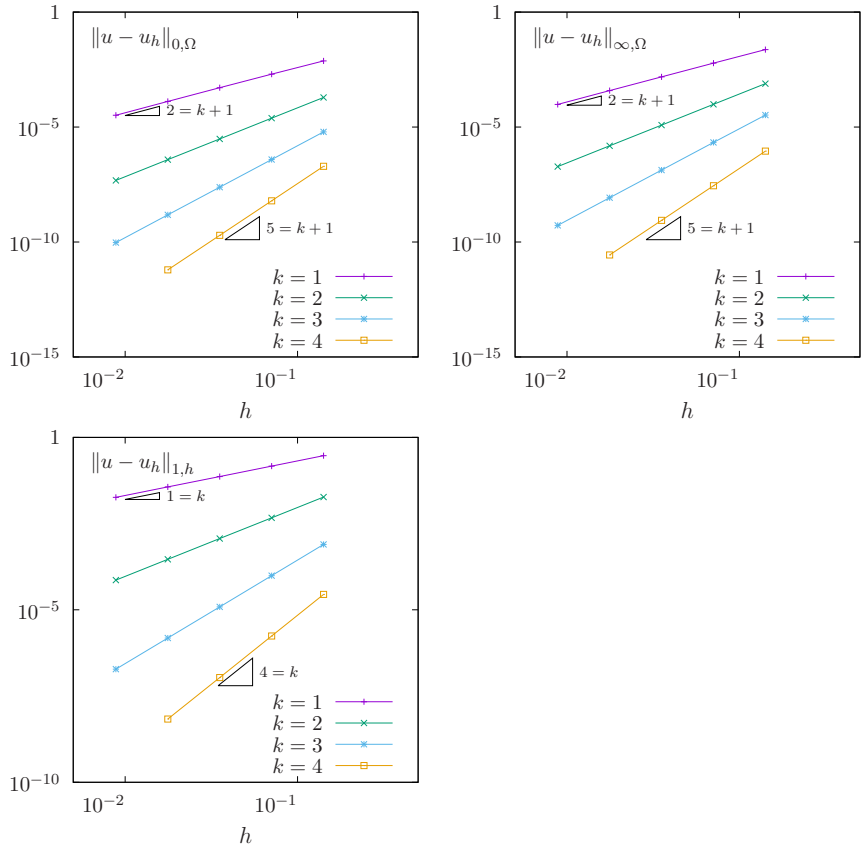


Figure 2.2: The discontinuous Galerkin method for the Poisson problem: convergence when $d = 2$.

where f and g are given. We introduce the folowing bilinear form $a_h(.,.)$ and linear for $l_h(.)$, defined for all $u, v \in H^1(\mathcal{T}_h)$ by (see e.g. [11, p. 127], eqn. (4.16)):

$$a_h(u,v) = \int_\Omega (u\,v + \nabla_h u.\nabla_h v)\,\mathrm{d}x \tag{2.3}$$

$$+ \sum_{S\in\mathscr{S}_h^{(i)}} \int_S (\beta\varpi_s\,[\![u]\!]\,[\![v]\!] - \{\!\{\nabla_h u.\mathbf{n}\}\!\}\,[\![v]\!] - [\![u]\!]\,\{\!\{\nabla_h v.\mathbf{n}\}\!\})\,\mathrm{d}s \tag{2.4}$$

$$l_h(v) = \int_\Omega f\,u\,\mathrm{d}x + \int_{\partial\Omega} g\,v\,\mathrm{d}s \tag{2.5}$$

Let us comment the changes between these forms and those used for the Poisson problem with Dirichlet boundary conditions. The Poisson operator $-\Delta$ has been replaced by the Helmholtz one $I-\Delta$ in order to have an unique solution. Remark also that the sum is performed in (2.1) for all internal sides in $\mathscr{S}_h^{(i)}$, while, in (2.1), for Dirichlet boundary conditions, it was for all sides in $\mathscr{S}_h$, i.e. for both boundary and internal sides. Also, the right-hand-side linear form $l_h(.)$. do no more involves any sum over sides.

Finally, the discrete variational formulation writes:

$(FV)_h$: *find* $u_h \in X_h$ *such that*

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

The following code implement this problem in the **Rheolef** environment.

Example file 2.2: `neumann_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "sinusprod_helmholtz.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  size_t d = omega.dimension();
  size_t k = Xh.degree();
  Float beta = (k+1)*(k+d)/d;
  trial u (Xh); test v (Xh);
  form  a = integrate (u*v + dot(grad_h(u),grad_h(v)))
          + integrate ("internal_sides",
                 beta*penalty()*jump(u)*jump(v)
               - jump(u)*average(dot(grad_h(v),normal()))
               - jump(v)*average(dot(grad_h(u),normal())));
  field lh = integrate (f(d)*v) + integrate ("boundary", g(d)*v);
  solver sa (a.uu());
  field uh(Xh);
  uh.set_u() = sa.solve(lh.u());
  dout << uh;
}
```

Comments

The right-hand side f and g are given by (2.2), volume 1, page 28. In that case, the exact solution is known. Running the program is obtained from the non-homogeneous Dirichlet case, by replacing `dirichlet_dg` by `neumann_dg`.

2.3 Nonlinear scalar hyperbolic problems with diffusion

A time-dependent nonlinear second order problem with nonlinear first order dominated terms problem writes:

(P): *find u, defined in $]0,T[\times\Omega$, such that*

$$\frac{\partial u}{\partial t} + \operatorname{div}\mathbf{f}(u) - \varepsilon\Delta u = 0 \quad \text{in }]0,T[\times\Omega \tag{2.6a}$$

$$u(t=0) = u_0 \quad \text{in } \Omega \tag{2.6b}$$

$$u = g \text{ on }]0,T[\times\partial\Omega \tag{2.6c}$$

where $\varepsilon > 0$, $T > 0$, $\Omega \subset \mathbb{R}^d$, $d = 1,2,3$ and the initial condition $\mathbf{u}_0$ being known. The function $\mathbf{f} : \mathbb{R} \longrightarrow \mathbb{R}^d$ is also known and supposed to be continuously differentiable. The initial data u_0, defined in Ω, and the boundary one, g, defined on $\partial\Omega$ are given.

Comparing (2.6a)-(2.6c) with the non-diffusive case (1.1a)-(1.1c) page 10, the second order term has been added in (2.6a) and the upstrem boundary condition has been replaced by a Dirichlet one (2.6c).

2.4 Example: the Burgers equation with diffusion

2.4.1 Problem statement and its exact solution

Our model problem in this chapter is the one-dimensional Burgers equation. It was introduced in section 1.3, page 14 with the choice $\mathbf{f}(u) = u^2/2$, for all $u \in \mathbb{R}$. Equation (2.6a) admits an exact

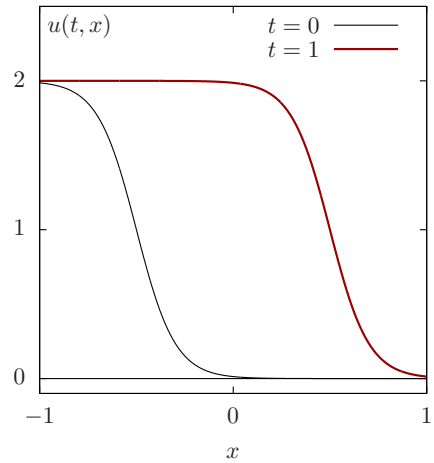


Figure 2.3: An exact solution for the Burgers equation with diffusion ($\varepsilon = 10^{-1}$, $x_0 = -1/2$).

solution

$$u(t,x) \quad = \quad 1 - \tanh\left(\frac{x - x_0 - t}{2\varepsilon}\right) \tag{2.7}$$

Example file 2.3: `burgers_diffusion_exact.icc`

```
struct u_exact {
  Float operator() (const point& x) const {
    return 1 - tanh((x[0]-x0-t)/(2*epsilon)); }
  u_exact (Float e1, Float t1=0) : epsilon(e1), t(t1), x0(-0.5) {}
  Float M() const { return 0; }
  Float epsilon, t, x0;
};
using u_init = u_exact;
using g = u_exact;
```

The solution is represente on Fig. 2.3. Here x_0 represent the position of the front at $t = 0$ and ε is a characteristic width of the front. The initial and boundary condition are choosen such that $u(t,x)$ is the solution of (2.6a)-(2.6c).

Fig. 2.4.a plots the error versus Δt for the semi-implicit scheme when $k = 1$ and 2, and for $h = 2/50$. The time step for which the error becomes independent upon Δt and depends only upon h is of about $\Delta t = 10^{-3}$ when $k = 1$ and of about 10^{-5} when $k = 2$. This approach is clearly inefficient for hight order polynomial k and a hiher order time scheme is required.

Fig. 2.4.b plots the error versus Δt for the Runge-Kutta semi-implicit scheme with $p = 3$, $k = 1$ and $h = 2/200$. The scheme is clearly only first-order, which is still unexpected. More work is required...

2.4.2 Space discretization

The *discontinuous* finite element space is defined by:

$$X_h = \{v_h \in L^2(\Omega); v_{h|K} \in P_k,\ \forall K \in \mathcal{T}_h\}$$

where $k \geqslant 1$ is the polynomial degree. As elements of X_h do not belongs to $H^1(\Omega)$, due to discontinuities at inter-elements, we introduce the broken Sobolev space:

$$H^1(\mathcal{T}_h) = \{v \in L^2(\Omega);\ v_{|K} \in H^1(K),\ \forall K \in \mathcal{T}_h\}$$

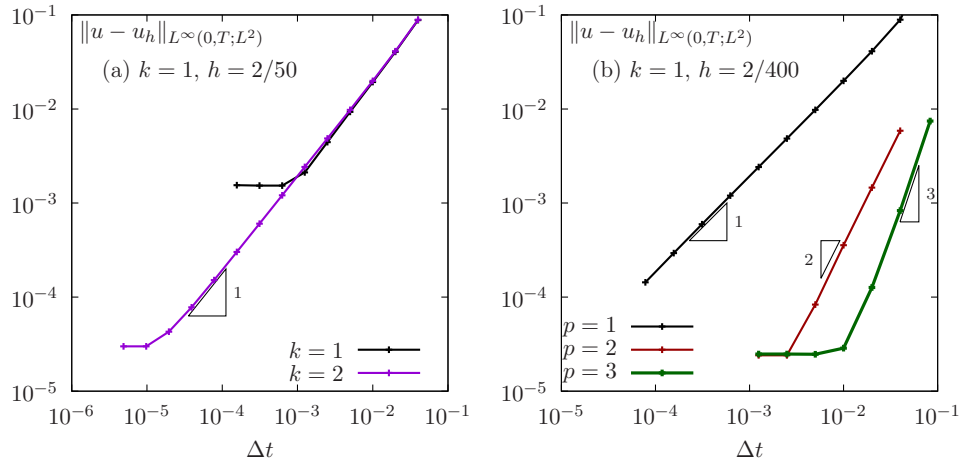


Figure 2.4: Convergence of the first order semi-implicit scheme for the Burgers equation with diffusion ($\epsilon = 0.1$, $T = 1$). (a) first order semi-implicit scheme ; (b) Runge-Kutta semi-implicit scheme with $p = 3$.

such that $X_h \subset H^1(\mathcal{T}_h)$. As for the Dirichlet problem, introduce the folowing bilinear form $a_h(.,.)$ and linear for $l_h(.)$, defined for all $u, v \in H^1(\mathcal{T}_h)$ by (see e.g. [11, p. 125 and 127], eqn. (4.12)):

$$a_h(u,v) = \int_\Omega \nabla_h u.\nabla_h v \,\mathrm{d}x + \sum_{S\in\mathscr{S}_h} \int_S (\eta_s \,[\![u]\!]\,[\![v]\!] - \{\!\{\nabla_h u.\mathbf{n}\}\!\}\,[\![v]\!] - [\![u]\!]\,\{\!\{\nabla_h v.\mathbf{n}\}\!\})\,\mathrm{d}s \quad (2.8)$$

$$\ell_h(v) = \int_{\partial\Omega} (\eta_s\, g\, v - g\,\nabla_h v.\mathbf{n})\,\mathrm{d}s \quad (2.9)$$

The semi-discrete problem writes in variational form [11, p. 100]:

$(P)_h$: *find* $u_h \in C^1([0,T], X_h)$ *such that*

$$\begin{aligned} \int_\Omega \frac{\partial u_h}{\partial t} v_h \,\mathrm{d}x \int_\Omega G_h(u_h)\, v_h \,\mathrm{d}x + \varepsilon\, a_h(u_h, v_h) &= \varepsilon\, \ell_h(v_h), \quad \forall v_h \in X_h \\ u_h(t{=}0) &= \pi_h(u_0) \end{aligned}$$

where G_h has been introduced in (1.2), page 11.

2.4.3 Time discretization

Explicit Runge-Kutta scheme is possible for this problem but it leads to an excessive Courant-Friedrichs-Levy condition for the time step Δt, that is required to be lower than an upper bound that varies in $\mathcal{O}(h^2)$. The idea here is to continue to explicit the first order nonlinearr terms and implicit the linear second order terms. Semi-implicit second order Runge-Kutta scheme was first introduced in 1997 by Ascher, Ruuth and Spiteri [1] and then extended in 2001 to third and fourth order by Calvo, de Frutos and Novo [2]. In 2015, Wang, Shu and Zhang [24, 25] applied it in the context of the discontinuous Galerkin method. The finite dimensional problem can be rewritten as

$(P)_h$: *find* $u_h \in C^1([0,T], X_h)$ *such that*

$$\begin{aligned} \frac{\partial u_h}{\partial t} + G_h(t, u_h) + A_h(t, u_h) &= 0, \quad \forall t \in]0,T[\\ u_h(t{=}0) &= \pi_h(u_0) \end{aligned}$$

where G_h has been introduced in (1.2), page 11 and A_h denotes the diffusive term. The semi-implicit Runge-Kutta scheme with $p \geqslant 0$ intermediate steps writes at time step t_n:

$$u_h^{n,0} = u_h^n \tag{2.10a}$$

$$u_h^{n,i} = u_h^n - \Delta t \sum_{j=1}^{i} \alpha_{i,j} A_h\left(t_{n,j},\, u_h^{n,j}\right) - \Delta t \sum_{j=0}^{i-1} \tilde{\alpha}_{i,j} G_h\left(t_{n,j},\, u_h^{n,j}\right), \quad i = 1, \ldots, p \tag{2.10b}$$

$$u_h^{n+1} = u_h^n - \Delta t \sum_{i=1}^{p} \beta_i A_h\left(t_{n,i},\, u_h^{n,i}\right) - \Delta t \sum_{i=0}^{p} \tilde{\beta}_i G_h\left(t_{n,i},\, u_h^{n,i}\right) \tag{2.10c}$$

where $\left(u_h^{n,i}\right)_{1 \leqslant i \leqslant p}$ are the $p \geqslant 1$ intermediate states, $t_{n,i} = t_n + \gamma_i \Delta t$, $\gamma_i = \sum_{j=1}^{i} \alpha_{i,j} = \sum_{j=0}^{i-1} \tilde{\alpha}_{i,j}$, and $(\alpha_{i,j})_{0 \leqslant i,j \leqslant p}$, $(\tilde{\alpha}_{i,j})_{0 \leqslant i,j \leqslant p}$, $(\beta_i)_{0 \leqslant i \leqslant p}$ and $(\tilde{\beta}_i)_{0 \leqslant i \leqslant p}$ are the coefficients of the scheme [1,2,25]. At each time step, have to solve p linear systems. From (2.10b) we get for all $i = 1, \ldots, p$:

$$\left(I + \Delta t\, \alpha_{i,i} A_h\left(t_{n,i}\right)\right) u_h^{n,i} = u_h^n - \Delta t \sum_{j=1}^{i-1} \alpha_{i,j} A_h\left(t_{n,j},\, u_h^{n,j}\right) - \Delta t \sum_{j=0}^{i-1} \tilde{\alpha}_{i,j} G_h\left(t_{n,j},\, u_h^{n,j}\right)$$

Notice that when the matrix coefficients of $A_h(t,.)$ are indepencdent of t, the matrix involved on the right-hand-side of the previous equation can be factored one time for all.

Example file 2.4: `burgers_diffusion_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "burgers.icc"
#include "burgers_flux_godunov.icc"
#include "runge_kutta_semiimplicit.icc"
#include "burgers_diffusion_exact.icc"
#undef  NEUMANN
#include "burgers_diffusion_operators.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2]);
  size_t k = Xh.degree();
  Float epsilon = (argc > 3) ? atof(argv[3]) : 0.1;
  size_t nmax   = (argc > 4) ? atoi(argv[4]) : 500;
  Float  tf     = (argc > 5) ? atof(argv[5]) : 1;
  size_t p      = (argc > 6) ? atoi(argv[6]) : min(k+1,rk::pmax);
  Float delta_t = tf/nmax;
  size_t d = omega.dimension();
  Float beta = (k+1)*(k+d)/d;
  trial u (Xh); test v (Xh);
  form m = integrate (u*v);
  form_option_type fopt;
  fopt.invert = true;
  form inv_m = integrate (u*v, fopt);
  form a = epsilon*(
              integrate (dot(grad_h(u),grad_h(v)))
#ifdef  NEUMANN
            + integrate ("internal_sides",
#else //  NEUMANN
            + integrate ("sides",
#endif //  NEUMANN
                  beta*penalty()*jump(u)*jump(v)
                - jump(u)*average(dot(grad_h(v),normal()))
                - jump(v)*average(dot(grad_h(u),normal()))));
  vector<solver> sc (p+1);
  for (size_t i = 1; i <= p; ++i) {
    form ci = m + delta_t*rk::alpha[p][i][i]*a;
    sc[i] = solver(ci.uu());
  }
  vector<field> uh(p+1, field(Xh,0));
  uh[0] = interpolate (Xh, u_init(epsilon));
  branch even("t","u");
  dout << catchmark("epsilon") << epsilon << endl
       << even(0,uh[0]);
  for (size_t n = 0; n < nmax; ++n) {
    Float tn = n*delta_t;
    Float t  = tn + delta_t;
    field uh_next = uh[0] - delta_t*rk::tilde_beta[p][0]*(inv_m*gh(epsilon, tn, uh[0], v));
    for (size_t i = 1; i <= p; ++i) {
      Float ti = tn + rk::gamma[p][i]*delta_t;
      field rhs = m*uh[0] - delta_t*rk::tilde_alpha[p][i][0]*gh(epsilon, tn, uh[0], v);
      for (size_t j = 1; j <= i-1; ++j) {
        Float tj = tn + rk::gamma[p][j]*delta_t;
        rhs -= delta_t*( rk::alpha[p][i][j]*(a*uh[j] - lh(epsilon,tj,v))
                       + rk::tilde_alpha[p][i][j]*gh(epsilon, tj, uh[j], v));
      }
      rhs += delta_t*rk::alpha[p][i][i]*lh (epsilon, ti, v);
      uh[i].set_u() = sc[i].solve (rhs.u());
      uh_next -= delta_t*(inv_m*( rk::beta[p][i]*(a*uh[i] - lh(epsilon,ti,v))
                                + rk::tilde_beta[p][i]*gh(epsilon, ti, uh[i], v)));
    }
    uh_next = limiter(uh_next);
    dout << even(tn+delta_t,uh_next);
    uh[0] = uh_next;
  }
}
```

Example file 2.5: `burgers_diffusion_operators.icc`

```
field lh (Float epsilon, Float t, const test& v) {
#ifdef  NEUMANN
  return field (v.get_vf_space(), 0);
#else //  NEUMANN
  size_t d = v.get_vf_space().get_geo().dimension();
  size_t k = v.get_vf_space().degree();
  Float beta = (k+1)*(k+d)/d;
  return epsilon*integrate ("boundary",
         beta*penalty()*g(epsilon,t)*v
       - g(epsilon,t)*dot(grad_h(v),normal()));
#endif //  NEUMANN
}
field gh (Float epsilon, Float t, const field& uh, const test& v) {
  return - integrate (dot(compose(f,uh),grad_h(v)))
         + integrate ("internal_sides",
             compose (phi, normal(), inner(uh), outer(uh))*jump(v))
         + integrate ("boundary",
             compose (phi, normal(), uh, g(epsilon,t))*v);
}
```

Running the program

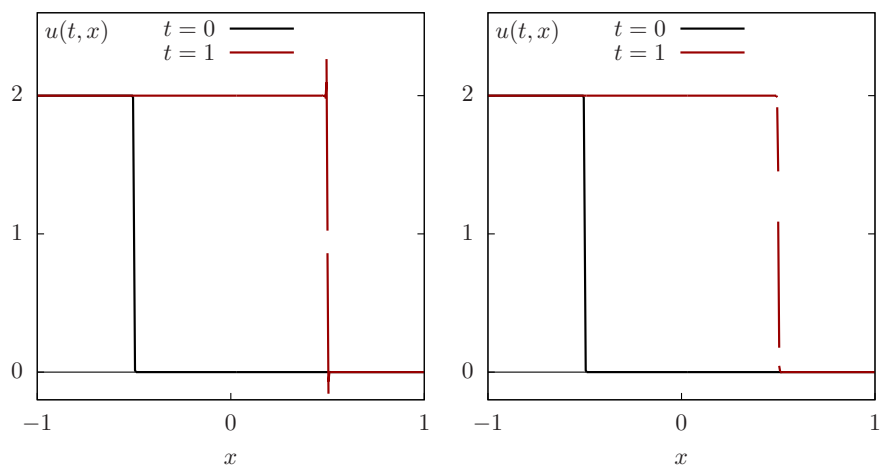


Figure 2.5: Burgers equation with a small diffusion ($\varepsilon = 10^{-3}$). Third order in time semi-implicit scheme with P_{1d} element. (left) without limiter ; (right) with limiter.

Running the program writes with $h = 2/400$ and $\varepsilon = 10^{-2}$ writes:

```
make burgers_diffusion_dg
mkgeo_grid -e 400 -a -1 -b 1 > line.geo
./burgers_diffusion_dg line P1d 0.01 1000 1 3 > line.branch
branch -gnuplot line.branch -umin -0.1 -umax 2.1
```

Decreasing $\varepsilon = 10^{-3}$ leads to a sharper solution:

```
./burgers_diffusion_dg line P1d 0.001 1000 1 3 > line.branch
branch -gnuplot line.branch -umin -0.1 -umax 2.1
```

As mentioned in [25], the time step should be chosen smaller when ε decreases. The result is shown on Fig. 2.5.left. Observe the oscillations near the smoothed shock when there is no limiter while the value goes outside $[0, 2]$. Conversely, with a limiter (see Fig. 2.5.right) the approximate solution is decreasing and there is no more oscilattions: the values remains in the range $[0 : 2]$.

Part II

Fluids and solids computations

Chapter 3

The linear elasticity and the Stokes problems

3.1 The linear elasticity problem

The elasticity problem (4.2) has been introduced in volume 1, section 4.1, page 51.

(P): *find* $\mathbf{u}$ *such that*

$$\begin{aligned} -\mathbf{div}\,(\lambda \mathrm{div}(\mathbf{u}).I + 2D(\mathbf{u})) &= \mathbf{f} \ \text{ in } \ \Omega \\ \mathbf{u} &= \mathbf{g} \ \text{ on } \ \partial\Omega \end{aligned}$$

where $\lambda \geqslant -1$ is a constant and $\mathbf{f}, \mathbf{g}$ given. This problem is a natural extension to vector-valued field of the Poisson problem with Dirichlet boundary conditions.

The variational formulation writes:

$(FV)_h$: *find* $\mathbf{u} \in \mathbf{V}(\mathbf{g})$ *such that*

$$a(\mathbf{u}, \mathbf{v}) = l_h(\mathbf{v}), \ \forall \mathbf{v} \in \mathbf{V}(0)$$

where

$$\begin{aligned} \mathbf{V}(\mathbf{g}) &= \{\mathbf{v} \in H^1(\Omega)^d; \ \mathbf{v} = \mathbf{g} \ \text{ on } \ \partial\Omega\} \\ a(\mathbf{u}, \mathbf{v}) &= \int_\Omega (\lambda \,\mathrm{div}(\mathbf{u})\,\mathrm{div}(\mathbf{v}) + 2D(\mathbf{u}) : D(\mathbf{v}))\ \mathrm{d}x \\ l(\mathbf{v}) &= \int_\Omega \mathbf{f}.\mathbf{v}\,\mathrm{d}x \end{aligned}$$

The discrete variational formulation writes:

$(FV)_h$: *find* $u_h \in \mathbf{X}_h$ *such that*

$$a_h(u_h, v_h) = l_h(v_h), \ \forall v_h \in \mathbf{X}_h$$

where

$$\begin{aligned} \mathbf{X}_h &= \{\mathbf{v}_h \in L^2(\Omega)^d; \mathbf{v}_{h|K} \in P_k^d, \ \forall K \in \mathcal{T}_h\} \\ a_h(\mathbf{u}, \mathbf{v}) &= \int_\Omega (\lambda \,\mathrm{div}_h(\mathbf{u})\,\mathrm{div}_h(\mathbf{v}) + 2D_h(\mathbf{u}) : D_h(\mathbf{v}))\ \mathrm{d}x \\ &+ \sum_{S \in \mathscr{S}_h} \int_S (\beta \varpi_s [\![\mathbf{u}]\!].[\![\mathbf{v}]\!] - [\![\mathbf{u}]\!].\{\!\{\lambda \mathrm{div}_h(\mathbf{v})\mathbf{n} + 2D_h(\mathbf{v})\mathbf{n}\}\!\} - [\![\mathbf{v}]\!].\{\!\{\lambda \mathrm{div}_h(\mathbf{u})\mathbf{n} + 2D_h(\mathbf{u})\mathbf{n}\}\!\})\ \mathrm{d}s \\ l_h(\mathbf{v}) &= \int_\Omega \mathbf{f}.\mathbf{v}\,\mathrm{d}x + \int_{\partial\Omega} \mathbf{g}.\,(\beta \varpi_s \mathbf{v} - \lambda \mathrm{div}_h(\mathbf{v})\mathbf{n} - 2D_h(\mathbf{v})\mathbf{n})\ \mathrm{d}s \end{aligned}$$

where $k \geqslant 1$ is the polynomial degree in $\mathbf{X}_h$.

Example file 3.1: `elasticity_taylor_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "taylor.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2], "vector");
  Float lambda = (argc > 3) ? atof(argv[3]) : 1;
  size_t d = omega.dimension();
  size_t k = Xh.degree();
  Float beta = (k+1)*(k+d)/d;
  trial u (Xh); test v (Xh);
  form  a = integrate (lambda*div_h(u)*div_h(v) + 2*ddot(Dh(u),Dh(v)))
          + integrate (omega.sides(),
                  beta*penalty()*dot(jump(u),jump(v))
                - lambda*dot(jump(u),average(div_h(v)*normal()))
                - lambda*dot(jump(v),average(div_h(u)*normal()))
                - 2*dot(jump(u),average(Dh(v)*normal()))
                - 2*dot(jump(v),average(Dh(u)*normal())));
  field lh = integrate (dot(f(),v))
           + integrate (omega.boundary(),
                  beta*penalty()*dot(g(),jump(v))
                - lambda*dot(g(),average(div_h(v)*normal()))
                - 2*dot(g(),average(Dh(v)*normal())));
  solver sa (a.uu());
  field uh(Xh);
  uh.set_u() = sa.solve(lh.u());
  dout << uh;
}
```

Comments

The data are given when $d = 2$ by:

$$\mathbf{g}(x) = \begin{pmatrix} -\cos(\pi x_0)\sin(\pi x_1) \\ \sin(\pi x_0)\cos(\pi x_1) \end{pmatrix} \quad \text{and} \quad \mathbf{f} = 2\pi^2\mathbf{g} \tag{3.1}$$

This choice is convenient since the exact solution is known $\mathbf{u} = \mathbf{g}$. This benmark solution was proposed in 1923 by Taylor [23] in the context of the Stokes problem. Notice that the solution is independent of λ since $\mathrm{div}(\mathbf{u}) = 0$.

Example file 3.2: `taylor.icc`

```
struct g {
  point operator() (const point& x) const {
    return point(-cos(pi*x[0])*sin(pi*x[1]),
                 sin(pi*x[0])*cos(pi*x[1])); }
  g() : pi(acos(Float(-1.0))) {}
  const Float pi;
};
struct f {
  point operator() (const point& x) const { return 2*sqr(pi)*_g(x); }
  f() : pi(acos(Float(-1.0))), _g() {}
  const Float pi; g _g;
};
```

As the exact solution is known, the error can be computed. The code code `elasticity_taylor_error_dg.cc` compute the error in L^2, L^∞ and energy norms. This code it is not listed here but is available in the **Rheolef** example directory. The computation writes:

```
make elasticity_taylor_dg elasticity_taylor_error_dg
```

```
mkgeo_grid -t 10 > square.geo
./elasticity_taylor_dg square P1d | ./elasticity_taylor_error_dg
./elasticity_taylor_dg square P2d | ./elasticity_taylor_error_dg
```

3.2 The Stokes problem

Let us consider the Stokes problem for the driven cavity in $\Omega =]0,1[^d$, $d = 2,3$. The problem has been introduced in volume 1, section 4.4, page 62.

(P): *find* $\mathbf{u}$ *and* p*, defined in* Ω*, such that*

$$\begin{array}{rcrcl} -\,\mathbf{div}(2D(\mathbf{u})) & + & \boldsymbol{\nabla} p & = & \mathbf{f} \text{ in } \Omega, \\ -\,\mathrm{div}\,\mathbf{u} & & & = & 0 \text{ in } \Omega, \\ \mathbf{u} & & & = & \mathbf{g} \text{ on } \partial\Omega \end{array}$$

where $\mathbf{f}$ and $\mathbf{g}$ are given. This problem is the extension to divergence free vector fields of the elasticity problem. The variational formulation writes:

$(VF)_h$ *find* $\mathbf{u} \in \mathbf{V}(\mathbf{g})$ *and* $p \in L^2(\Omega)$ *such that:*

$$\begin{array}{rcrcl} a(\mathbf{u},\mathbf{v}) & + & b(\mathbf{v},p) & = & l(\mathbf{v}),\ \forall \mathbf{v} \in \mathbf{V}(0), \\ b(\mathbf{u},q) & & & = & 0,\ \forall q \in L^2(\Omega) \end{array} \tag{3.2}$$

where

$$\begin{array}{rcl} \mathbf{V}(\mathbf{g}) & = & \{\mathbf{v} \in H^1(\Omega)^d;\ \mathbf{v} = \mathbf{g} \ \text{ on } \ \partial\Omega\} \\ a(\mathbf{u},\mathbf{v}) & = & \displaystyle\int_\Omega 2D(\mathbf{u}) : D(\mathbf{v})\,\mathrm{d}x \\ b(\mathbf{u},q) & = & \displaystyle -\int_\Omega \mathrm{div}(\mathbf{u})\,q\,\mathrm{d}x \\ l(\mathbf{v}) & = & \displaystyle\int_\Omega \mathbf{f}.\mathbf{v}\,\mathrm{d}x \end{array}$$

The discrete variational formulation writes:

$(VF)_h$ *find* $\mathbf{u}_h \in \mathbf{X}_h$ *and* $p_h \in Q_h$ *such that:*

$$\begin{array}{rcrcl} a_h(\mathbf{u}_h,\mathbf{v}_h) & + & b_h(\mathbf{v}_h,p_h) & = & l_h(\mathbf{v}_h),\ \forall \mathbf{v}_h \in \mathbf{X}_h, \\ b_h(\mathbf{u}_h,q_h) & - & c_h(p_h,q_h) & = & k_h(q),\ \forall q_h \in Q_h. \end{array} \tag{3.3}$$

The discontinuous finite element spaces are defined by:

$$\begin{array}{rcl} \mathbf{X}_h & = & \{\mathbf{v}_h \in L^2(\Omega)^d; \mathbf{v}_{h|K} \in P_k^d,\ \forall K \in \mathcal{T}_h\} \\ Q_h & = & \{q_h \in L^2(\Omega)^d; q_{h|K} \in P_k^d,\ \forall K \in \mathcal{T}_h\} \end{array}$$

where $k \geqslant 1$ is the polynomial degree. Notice that velocity and presure are approximated by the same polynomial order. This method was introduced by [9] and some recent theoretical results can be founded in [10]. The forms are defined for all $u, v \in H^1(\mathcal{T}_h)^d$ and $q \in L^2(\Omega)$ by (see

e.g. [11, p. 249]):

$$
\begin{aligned}
a_h(\mathbf{u},\mathbf{v}) &= \int_\Omega 2D_h(\mathbf{u}):D_h(\mathbf{v})\,\mathrm{d}x \\
&\quad + \sum_{S\in\mathscr{S}_h}\int_S \left(\beta\varpi_s[\![\mathbf{u}]\!].[\![\mathbf{v}]\!] - [\![\mathbf{u}]\!].\{\!\{2D_h(\mathbf{v})\mathbf{n}\}\!\} - [\![\mathbf{v}]\!].\{\!\{2D_h(\mathbf{u})\mathbf{n}\}\!\}\right)\,\mathrm{d}s \\
b_h(\mathbf{u},q) &= \int_\Omega \mathbf{u}.\nabla_h q\,\mathrm{d}x - \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \{\!\{\mathbf{u}\}\!\}.\mathbf{n}\,[\![q]\!]\,\mathrm{d}s \\
c_h(p,q) &= \sum_{S\in\mathscr{S}_h^{(i)}}\int_S h_s\,[\![p]\!]\,[\![q]\!]\,\mathrm{d}s \\
l_h(\mathbf{v}) &= \int_\Omega \mathbf{f}.\mathbf{v}\,\mathrm{d}s + \int_{\partial\Omega}\mathbf{g}.\left(\beta\varpi_s\,\mathbf{v} - 2D_h(\mathbf{v})\,\mathbf{n}\right)\,\mathrm{d}s \\
k_h(q) &= \int_{\partial\Omega}\mathbf{g}.\mathbf{n}\,q\,\mathrm{d}s
\end{aligned}
$$

The stabilization form c_h controls the pressure jump accross internal sides. This stabilization term is necessary when using equal order polynomial approximation for velocity and pressure. The definition of the forms is grouped in a subroutine: it will be reused later for the Navier-Stokes problem.

Example file 3.3: `stokes_dirichlet_dg.icc`

```
void stokes_dirichlet_dg (const space& Xh, const space& Qh,
  form& a, form& b, form& c, form& mp, field& lh, field& kh,
  quadrature_option_type qopt = quadrature_option_type())
{
  size_t k = Xh.degree();
  size_t d = Xh.get_geo().dimension();
  Float beta = (k+1)*(k+d)/d;
  trial u (Xh), p (Qh);
  test  v (Xh), q (Qh);
  a  = integrate (2*ddot(Dh(u),Dh(v)), qopt)
     + integrate ("sides",    beta*penalty()*dot(jump(u),jump(v))
                 - 2*dot(jump(u),average(Dh(v)*normal()))
                 - 2*dot(jump(v),average(Dh(u)*normal())), qopt);
  lh = integrate (dot(f(),v), qopt)
     + integrate ("boundary", beta*penalty()*dot(g(),v)
                - 2*dot(g(),Dh(v)*normal()), qopt);
  b  = integrate (dot(u,grad_h(q)), qopt)
     + integrate ("internal_sides", - dot(average(u),normal())*jump(q), qopt);
  kh = integrate ("boundary", dot(g(),normal())*q, qopt);
  c  = integrate ("internal_sides", h_local()*jump(p)*jump(q), qopt);
  mp = integrate (p*q, qopt);
}
```

A simple test program writes:

Example file 3.4: `stokes_taylor_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "taylor.icc"
#include "stokes_dirichlet_dg.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2], "vector");
  space Qh (omega, argv[2]);
  form a, b, c, mp;
  field lh, kh;
  stokes_dirichlet_dg (Xh, Qh, a, b, c, mp, lh, kh);
  field uh (Xh, 0), ph (Qh, 0);
  solver_abtb stokes (a.uu(), b.uu(), c.uu(), mp.uu());
  stokes.solve (lh.u(), kh.u(), uh.set_u(), ph.set_u());
  dout << catchmark("u")  << uh
       << catchmark("p")  << ph;
}
```

Comments

The data are given when $d = 2$ by (3.1). This choice is convenient since the exact solution is known $\mathbf{u} = \mathbf{g}$ and $p = 0$. The code `stokes_taylor_error_dg.cc` compute the error in L^2, L^∞ and energy norms. This code it is not listed here but is available in the **Rheolef** example directory. The computation writes:

```
make stokes_taylor_dg stokes_taylor_error_dg
mkgeo_grid -t 10 > square.geo
./stokes_taylor_dg square P1d | ./stokes_taylor_error_dg
./stokes_taylor_dg square P2d | ./stokes_taylor_error_dg
```

3.3 The stationnary Navier-Stokes problem

3.3.1 Problem statemment

The Navier-Stokes problem has been already introduced in volume 1, section 3.3 page 37. Here we consider the stationnary version of this problem. Let $Re \geqslant 0$ be the Reynolds number. The problem writes:

(P): *find* $\mathbf{u}$ *and* p*, defined in* Ω*, such that*

$$\begin{array}{rcrcrcl} Re\,(\mathbf{u}.\nabla)\mathbf{u} & - & \mathbf{div}(2D(\mathbf{u})) & + & \boldsymbol{\nabla} p & = & \mathbf{f} \text{ in } \Omega, \\ & & -\operatorname{div}\mathbf{u} & & & = & 0 \text{ in } \Omega, \\ & & \mathbf{u} & & & = & \mathbf{g} \text{ on } \partial\Omega \end{array}$$

Notice that, when $Re > 0$, the problem is nonlinear, due to the inertia term $\mathbf{u}.\nabla\mathbf{u}$. When $Re = 0$ the problem reduces to the linear Stokes problem, presented in the previous section/

The variationnal formulation of this nonlinear problem writes:

(FV): *find* $\mathbf{u} \in \mathbf{V}(\mathbf{g})$ *and* $p \in L^2(\Omega)$ *such that*

$$\begin{array}{rcrcrcl} Re\,t(\mathbf{u};\,\mathbf{u},\,\mathbf{v}) & + & a(\mathbf{u},\mathbf{v}) & + & b(\mathbf{v},p) & = & l(\mathbf{v}),\ \forall \mathbf{v} \in \mathbf{V}(0), \\ & & b(\mathbf{u},q) & & & = & 0,\ \forall q \in L^2(\Omega) \end{array}$$

where the space $\mathbf{V}(\mathbf{g})$ and forms a, b and l are given as in the previous section 3.2 for the Stokes problem and the trilinear form $t(.;.,.)$ is given by:

$$t(\mathbf{w};\,\mathbf{u},\,\mathbf{v}) \quad = \quad \int_\Omega ((\mathbf{w}.\nabla)\mathbf{u}).\mathbf{v}\,\mathrm{d}x$$

3.3.2 The discrete problem

Let

$$t(\mathbf{w};\, \mathbf{u},\, \mathbf{u}) \quad = \quad \int_\Omega (\mathbf{w}.\nabla\mathbf{u}).\mathbf{u}\,\mathrm{d}x$$

Observe that, for all $\mathbf{u},\, \mathbf{w} \in H^1(\Omega)^d$ we have

$$\begin{aligned}
\int_\Omega (\mathbf{w}.\nabla\mathbf{u}).\mathbf{u}\,\mathrm{d}x &= \sum_{i,j=0}^{d-1} \int_\Omega u_i\, w_j\, \partial_j(u_i)\,\mathrm{d}x \\
&= \sum_{i,j=0}^{d-1} -\int_\Omega u_i\, \partial_j(u_i\, w_j)\,\mathrm{d}x + \int_{\partial\Omega} u_i^2\, w_j\, n_j\,\mathrm{d}x \\
&= \sum_{i,j=0}^{d-1} -\int_\Omega u_i\, \partial_j(u_i)\, w_j\,\mathrm{d}x - \int_\Omega u_i^2\, \partial_j(w_j)\,\mathrm{d}x + \int_{\partial\Omega} u_i^2\, w_j\, n_j\,\mathrm{d}x \\
&= -\int_\Omega (\mathbf{w}.\nabla\mathbf{u}).\mathbf{u}\,\mathrm{d}x - \int_\Omega \mathrm{div}(\mathbf{w})\,|\mathbf{u}|^2\,\mathrm{d}x + \int_{\partial\Omega} \mathbf{w}.\mathbf{n}\,|\mathbf{u}|^2\,\mathrm{d}s \qquad (3.4)
\end{aligned}$$

Thus

$$t(\mathbf{w};\, \mathbf{u},\, \mathbf{u}) = \int_\Omega (\mathbf{w}.\nabla\mathbf{u}).\mathbf{u}\,\mathrm{d}x = -\frac{1}{2}\int_\Omega \mathrm{div}(\mathbf{w})\,|\mathbf{u}|^2\,\mathrm{d}x + \frac{1}{2}\int_{\partial\Omega} \mathbf{w}.\mathbf{n}\,|\mathbf{u}|^2\,\mathrm{d}s$$

When $\mathrm{div}(\mathbf{w}) = 0$, the trilinear form $t(.;.,.)$ reduces to a boundary term: it is formaly skew-symmetric. The skew-symmetry of t is an important property: let $(\mathbf{v}, q) = (\mathbf{u}, p)$ as test functions in (FV). We obtain:

$$a(\mathbf{u}, \mathbf{u}) = l(\mathbf{u})$$

In other words, we obtain the same energy balance as for the Stokes flow and inertia do not contribute to the energy balance. This is an important property and we aim at obtaining the same one at the discrete level. As the discrete solution $\mathbf{u}_h$ is not exactly divergence free, following Temam, we introduce the following modified trilinear form:

$$t^*(\mathbf{w};\, \mathbf{u},\, \mathbf{v}) \quad = \quad \int_\Omega \left((\mathbf{w}.\nabla\mathbf{u})\,.\mathbf{v} + \frac{1}{2}\mathrm{div}(\mathbf{w})\,\mathbf{u}.\mathbf{v} \right)\,\mathrm{d}x - \frac{1}{2}\int_{\partial\Omega} (\mathbf{w}.\mathbf{n})\,\mathbf{u}.\mathbf{v}\,\mathrm{d}s,\ \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in H^1(\Omega)^d$$

This form integrates the non-vanishing terms and we have:

$$t^*(\mathbf{w};\, \mathbf{u},\, \mathbf{u}) \quad = \quad 0,\ \forall \mathbf{u}, \mathbf{w} \in H^1(\Omega)^d$$

When the discrete solution is not exactly divergence free, it is better to use t^* than t.

The discontinuous finite element spaces $\mathbf{X}_h$ and Q_h and forms a_h, b_h, c_h, l_h and k_h are defined as in the previous section. Let us introduce t_h^*, the following discrete trilinear form, defined for all $\mathbf{u}_h, \mathbf{v}_h, \mathbf{w}_h \in \mathbf{X}_h$:

$$t_h^*(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) \quad = \quad \int_\Omega \left((\mathbf{w}_h.\nabla_h\mathbf{u}_h)\,.\mathbf{v}_h + \frac{1}{2}\mathrm{div}_h(\mathbf{w}_h)\,\mathbf{u}_h.\mathbf{v}_h \right)\,\mathrm{d}x - \frac{1}{2}\int_{\partial\Omega} (\mathbf{w}_h.\mathbf{n})\,\mathbf{u}_h.\mathbf{v}_h\,\mathrm{d}s$$

Notice that t_h^* is similar to t^*: the gradient and divergence has been replaced by their broken counterpart in the first term. As $\mathbf{X}_h \not\subset H^1(\Omega)^d$, the skew-symmetry property is not expected to be true at the discrete level. Then

$$t_h^*(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{u}_h) \quad = \quad \sum_{K\in\mathscr{T}_h} \int_K \left((\mathbf{w}_h.\nabla\mathbf{u}_h)\,.\mathbf{u}_h + \frac{1}{2}\mathrm{div}(\mathbf{w}_h)\,|\mathbf{u}_h|^2 \right)\,\mathrm{d}x - \frac{1}{2}\int_{\partial\Omega} (\mathbf{w}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s$$

As the restriction of $\mathbf{u}_h$ and $\mathbf{w}_h$ to each $K \in \mathscr{T}_h$ belongs to $H^1(K)^d$, we have, using a similar integration by part:

$$\int_K (\mathbf{w}_h.\nabla \mathbf{u}_h).\mathbf{u}_h \,\mathrm{d}x = -\frac{1}{2}\int_K \mathrm{div}(\mathbf{w}_h)\,|\mathbf{u}_h|^2\,\mathrm{d}x + \frac{1}{2}\int_{\partial K} (\mathbf{w}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s$$

Thus

$$t_h^*(\mathbf{w}_h;\,\mathbf{u}_h,\,\mathbf{u}_h) \quad = \quad \frac{1}{2}\sum_{K\in\mathscr{T}_h}\int_{\partial K} (\mathbf{w}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s - \frac{1}{2}\int_{\partial\Omega} (\mathbf{w}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s$$

The terms on boundary sides vanish while those on internal sides can be grouped:

$$t_h^*(\mathbf{w}_h;\,\mathbf{u}_h,\,\mathbf{u}_h) \quad = \quad \frac{1}{2}\sum_{S\in\mathscr{S}_h^{(i)}}\int_S [\![|\mathbf{u}_h|^2\mathbf{w}_h]\!].\mathbf{n}\,\mathrm{d}s$$

The jump term $[\![(\mathbf{u}_h.\mathbf{v}_h)\,\mathbf{w}_h]\!].\mathbf{n}$ is not easily manageable and could be developed. A short computation shows that, for all scalar fields ϕ, φ we have on any internal side:

$$[\![\phi\varphi]\!] \quad = \quad [\![\phi]\!]\{\!\{\varphi\}\!\} + \{\!\{\phi\}\!\}[\![\varphi]\!] \tag{3.5}$$

$$\{\!\{\phi\varphi\}\!\} \quad = \quad \{\!\{\phi\}\!\}\{\!\{\varphi\}\!\} + \frac{1}{4}[\![\phi]\!][\![\varphi]\!] \tag{3.6}$$

Then

$$\begin{aligned} t_h^*(\mathbf{w}_h;\,\mathbf{u}_h,\,\mathbf{u}_h) \quad &= \quad \frac{1}{2}\sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left(\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n}\,[\![|\mathbf{u}_h|^2]\!] + [\![\mathbf{w}_h]\!].\mathbf{n}\,\{\!\{|\mathbf{u}_h|^2\}\!\}\right)\,\mathrm{d}s \\ &= \quad \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left(\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n}\,([\![\mathbf{u}_h]\!].\{\!\{\mathbf{u}_h\}\!\}) + \frac{1}{2}[\![\mathbf{w}_h]\!].\mathbf{n}\,\{\!\{|\mathbf{u}_h|^2\}\!\}\right)\,\mathrm{d}s \end{aligned}$$

Thus, as expected, the skew-symmetry property is no more satisfied at the discrete level, due to the jumps of the fields at the inter-element boundaries. Following the previous idea, we introduce the following modified discrete trilinear form:

$$\begin{aligned} t_h(\mathbf{w}_h;\,\mathbf{u}_h,\,\mathbf{v}_h) \quad &= \quad t_h^*(\mathbf{w}_h;\,\mathbf{u}_h,\,\mathbf{v}_h) - \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left(\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n}\,([\![\mathbf{u}_h]\!].\{\!\{\mathbf{v}_h\}\!\}) + \frac{1}{2}[\![\mathbf{w}_h]\!].\mathbf{n}\,\{\!\{\mathbf{u}_h.\mathbf{v}_h\}\!\}\right)\,\mathrm{d}s \\ &= \quad \int_\Omega \left((\mathbf{w}_h.\nabla_h\mathbf{u}_h)\,.\mathbf{v}_h + \frac{1}{2}\mathrm{div}_h(\mathbf{w}_h)\,\mathbf{u}_h.\mathbf{v}_h\right)\,\mathrm{d}x - \frac{1}{2}\int_{\partial\Omega}(\mathbf{w}_h.\mathbf{n})\,\mathbf{u}_h.\mathbf{v}_h\,\mathrm{d}s \\ &\quad - \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left(\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n}\,([\![\mathbf{u}_h]\!].\{\!\{\mathbf{v}_h\}\!\}) + \frac{1}{2}[\![\mathbf{w}_h]\!].\mathbf{n}\,\{\!\{\mathbf{u}_h.\mathbf{v}_h\}\!\}\right)\,\mathrm{d}s \end{aligned} \tag{3.7}$$

This expression has been proposed by Pietro and Ern [10, p. 22], eqn (72) (see also [11, p. 272], eqn (6.57)). The boundary term introduced in t_h may be compensated in the right-hand side:

$$l_h^*(\mathbf{v}) \quad := \quad l_h(\mathbf{v}) - \frac{Re}{2}\int_{\partial\Omega}(\mathbf{g}.\mathbf{n})\,\mathbf{g}.\mathbf{v}_h\,\mathrm{d}s$$

Notice that the boundary term introduced in t_h is compensated in the right-hand side l_h^*.

Example file 3.5: `inertia.icc`

```
template<class W, class U, class V>
form inertia (W w, U u, V v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return
    integrate (dot(grad_h(u)*w,v) + 0.5*div_h(w)*dot(u,v), qopt)
  + integrate ("boundary", - 0.5*dot(w,normal())*dot(u,v), qopt)
  + integrate ("internal_sides",
      - dot(average(w),normal())*dot(jump(u),average(v))
      - 0.5*dot(jump(w),normal())
          *(dot(average(u),average(v)) + 0.25*dot(jump(u),jump(v))), qopt);
}
field inertia_fix_rhs (test v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return integrate("boundary", - 0.5*dot(g(),normal())*dot(g(),v), qopt);
}
```

The discrete problem is

$(FV)_h$: *find* $\mathbf{u}_h \in \mathbf{X}_h$ *and* $p \in Q_h$ *such that*

$$\begin{array}{rcrcrcl} Re\, t_h(\mathbf{u}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) & + & a_h(\mathbf{u}_h, \mathbf{v}_h) & + & b_h(\mathbf{v}_h, p_h) & = & l_h^*(\mathbf{v}_h),\ \forall \mathbf{v}_h \in \mathbf{X}_h, \\ & & b_h(\mathbf{u}_h, q_h) & - & c_h(p_h, q_h) & = & k_h(q),\ \forall q_h \in Q_h \end{array} \tag{3.8}$$

The simplest approach for solving the discrete problem is to consider a fixed-point algorithm. The sequence $\left(\mathbf{u}_h^{(k)}\right)_{k \geqslant 0}$ is defined by reccurence as:

- $k = 0$: let $\mathbf{u}_h^{(0)} \in \mathbf{X}_h$ being known.

- $k \geqslant 0$: let $\mathbf{u}_h^{(k-1)} \in \mathbf{X}_h$ given. Find $\mathbf{u}_h^{(k)} \in \mathbf{X}_h$ and $p_h^{(k)} \in Q_h$ such that

$$\begin{array}{rcrcrcl} Re\, t_h\left(\mathbf{u}_h^{(k-1)};\, \mathbf{u}_h^{(k)},\, \mathbf{v}_h\right) & + & a_h\left(\mathbf{u}_h^{(k)}, \mathbf{v}_h\right) & + & b_h\left(\mathbf{v}_h, p_h^{(k)}\right) & = & l_h^*(\mathbf{v}_h),\ \forall \mathbf{v}_h \in \mathbf{X}_h, \\ & & b_h\left(\mathbf{u}_h^{(k)}, q_h\right) & - & c_h\left(p_h^{(k)}, q_h\right) & = & k_h(q),\ \forall q_h \in Q_h. \end{array}$$

At each step $k \geqslant 0$, this algorithm involves a linear subproblem of Stokes-type.

Example file 3.6: `navier_stokes_taylor_dg.cc`

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "taylor.icc"
#include "stokes_dirichlet_dg.icc"
#include "inertia.icc"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  geo omega (argv[1]);
  space Xh (omega, argv[2], "vector");
  space Qh (omega, argv[2]);
  Float  Re       = (argc > 3) ? atof(argv[3]) : 1;
  size_t max_iter = (argc > 4) ? atoi(argv[4]) : 1;
  form a, b, c, mp;
  field lh, kh;
  stokes_dirichlet_dg (Xh, Qh, a, b, c, mp, lh, kh);
  field uh (Xh, 0), ph (Qh, 0);
  solver_abtb stokes (a.uu(), b.uu(), c.uu(), mp.uu());
  stokes.solve (lh.u(), kh.u(), uh.set_u(), ph.set_u());
  trial u (Xh); test v (Xh);
  form a1 = a + Re*inertia (uh, u, v);
  lh += Re*inertia_fix_rhs (v);
  derr << "#k r as" << endl;
  for (size_t k = 0; k < max_iter; ++k) {
    solver_abtb stokes (a1.uu(), b.uu(), c.uu(), mp.uu());
    stokes.solve (lh.u(), kh.u(), uh.set_u(), ph.set_u());
    form th = inertia (uh, u, v);
    a1 = a + Re*th;
    field rh = a1*uh + b.trans_mult(ph) - lh;
    derr << k << " " << rh.max_abs() << " " << th(uh,uh) << endl;
  }
  dout << catchmark("Re") << Re << endl
       << catchmark("u")  << uh
       << catchmark("p")  << ph;
}
```

Comments

The data are given when $d = 2$ by (3.1). This choice is convenient since the exact solution is known $\mathbf{u} = \mathbf{g}$ and $p = -(Re/4)(\cos(2\pi x_0) + \cos(2\pi x_1))$. The code `navier_stokes_taylor_error_dg.cc` compute the error in L^2, L^∞ and energy norms. This code it is not listed here but is available in the **Rheolef** example directory. The computation writes:

```
make navier_stokes_taylor_dg navier_stokes_taylor_error_dg
./navier_stokes_taylor_dg square P1d 10 10 | ./navier_stokes_taylor_error_dg
./navier_stokes_taylor_dg square P2d 10 10 | ./navier_stokes_taylor_error_dg
```

3.3.3 A conservative variant

Remark the identity

$$\mathbf{div}(\mathbf{u} \otimes \mathbf{u}) = (\mathbf{u}.\nabla)\mathbf{u} + \operatorname{div}(\mathbf{u})\,\mathbf{u}$$

The momentum conservation can be rewritten in conservative form and the problem writes:
($\tilde{P}$): *find* $\mathbf{u}$ *and* p*, defined in* Ω*, such that*

$$\begin{array}{rcrcl} \mathbf{div}(Re\,\mathbf{u} \otimes \mathbf{u} - 2D(\mathbf{u})) & + & \boldsymbol{\nabla} p & = & \mathbf{f} \text{ in } \Omega, \\ -\operatorname{div}\mathbf{u} & & & = & 0 \text{ in } \Omega, \\ \mathbf{u} & & & = & \mathbf{g} \text{ on } \partial\Omega \end{array}$$

Notice the Green formulae (see volume 1, appendix A.2, page 147):

$$\int_\Omega \mathbf{div}(\mathbf{u}\otimes\mathbf{u}).\mathbf{v}\,\mathrm{d}x = -\int_\Omega (\mathbf{u}\otimes\mathbf{u}):\nabla\mathbf{v}\,\mathrm{d}x + \int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,(\mathbf{u}.\mathbf{v})\,\mathrm{d}s$$

The variationnal formulation is:
($\widetilde{FV}$): *find* $\mathbf{u}\in\mathbf{V}(\mathbf{g})$ *and* $p\in L^2(\Omega)$ *such that*

$$\begin{array}{rcccccl} Re\,\tilde{t}(\mathbf{u};\,\mathbf{u},\,\mathbf{v}) & + & a(\mathbf{u},\mathbf{v}) & + & b(\mathbf{v},p) & = & \tilde{l}(\mathbf{v}),\ \forall\mathbf{v}\in\mathbf{V}(0), \\ & & b(\mathbf{u},q) & & & = & 0,\ \forall q\in L^2(\Omega) \end{array}$$

where the forms $\tilde{t}$ and $\tilde{l}_h$ are given by:

$$\begin{aligned} \tilde{t}(\mathbf{w};\,\mathbf{u},\,\mathbf{v}) &= -\int_\Omega (\mathbf{w}\otimes\mathbf{u}):\nabla\mathbf{v}\,\mathrm{d}x \\ \tilde{l}(\mathbf{v}) &= l(\mathbf{v}) - Re\int_{\partial\Omega}(\mathbf{g}.\mathbf{n})\,(\mathbf{g}.\mathbf{v})\,\mathrm{d}s \end{aligned}$$

Notice that the right-hand side $\tilde{l}$ contains an additional term that compensates those comming from the integration by parts. Then, with $\mathbf{v}=\mathbf{u}$:

$$\begin{aligned} \tilde{t}(\mathbf{w};\,\mathbf{u},\,\mathbf{u}) &= -\int_\Omega (\mathbf{w}\otimes\mathbf{u}):\nabla\mathbf{u}\,\mathrm{d}x \\ &= \int_\Omega \mathbf{div}(\mathbf{w}\otimes\mathbf{u}).\mathbf{u}\,\mathrm{d}x - \int_{\partial\Omega}(\mathbf{w}\otimes\mathbf{u}):(\mathbf{u}\otimes\mathbf{n})\,\mathrm{d}x \\ &= \int_\Omega \left(((\mathbf{u}.\nabla)\mathbf{w}).\mathbf{u} + \mathrm{div}(\mathbf{u})\,(\mathbf{u}.\mathbf{w})\right)\,\mathrm{d}x - \int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}x \end{aligned}$$

From an integration by part similar to (3.4):

$$\int_\Omega (\mathbf{u}.\nabla\mathbf{w}).\mathbf{u}\,\mathrm{d}x = -\int_\Omega (\mathbf{u}.\nabla\mathbf{u}).\mathbf{w}\,\mathrm{d}x - \int_\Omega \mathrm{div}(\mathbf{u})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}x + \int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}s$$

The term $(\mathbf{u}.\nabla\mathbf{w}).\mathbf{u}$ do not reapper after the integration by parts: instead, it appears $(\mathbf{u}.\nabla\mathbf{u}).\mathbf{w}$. Thus, the structure of the $\tilde{t}$ trilinear form do not permit a general skew-symmetry property as it was the case for t. It requires the three arguments to be the same:

$$\tilde{t}(\mathbf{u};\,\mathbf{u},\,\mathbf{u}) = \int_\Omega \left(((\mathbf{u}.\nabla)\mathbf{u}).\mathbf{u} + \mathrm{div}(\mathbf{u})\,|\mathbf{u}|^2\right)\,\mathrm{d}x - \int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,|\mathbf{u}|^2\,\mathrm{d}x$$

Using (3.4) with $\mathbf{w}=\mathbf{u}$ leads to:

$$\int_\Omega ((\mathbf{u}.\nabla)\mathbf{u}).\mathbf{u}\,\mathrm{d}x = -\frac{1}{2}\int_\Omega \mathrm{div}(\mathbf{u})\,|\mathbf{u}|^2\,\mathrm{d}x + \frac{1}{2}\int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,|\mathbf{u}|^2\,\mathrm{d}s \tag{3.9}$$

Then

$$\tilde{t}(\mathbf{u};\,\mathbf{u},\,\mathbf{u}) = \frac{1}{2}\int_\Omega \mathrm{div}(\mathbf{u})\,|\mathbf{u}|^2\,\mathrm{d}x - \frac{1}{2}\int_{\partial\Omega}(\mathbf{u}.\mathbf{n})\,|\mathbf{u}|^2\,\mathrm{d}s$$

When working with velocities that are not divergence-free, a possible modification of the trilinear form $\tilde{t}$ is to consider

$$\begin{aligned} \tilde{t}^*(\mathbf{w};\,\mathbf{u},\,\mathbf{v}) &= \tilde{t}(\mathbf{w};\,\mathbf{u},\,\mathbf{v}) - \frac{1}{2}\int_\Omega \mathrm{div}(\mathbf{v})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}x + \frac{1}{2}\int_{\partial\Omega}(\mathbf{v}.\mathbf{n})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}s \\ &= -\int_\Omega \left((\mathbf{w}\otimes\mathbf{u}):D(\mathbf{v}) + \frac{1}{2}\mathrm{div}(\mathbf{v})\,(\mathbf{u}.\mathbf{w})\right)\,\mathrm{d}x + \frac{1}{2}\int_{\partial\Omega}(\mathbf{v}.\mathbf{n})\,(\mathbf{u}.\mathbf{w})\,\mathrm{d}s \end{aligned}$$

Then we have

$$\tilde{t}^*(\mathbf{u};\, \mathbf{u},\, \mathbf{u}) \quad = \quad 0,\ \forall \mathbf{u} \in H^1(\Omega)^d$$

The new variationnal formulation is:
$(\widetilde{FV})^*$: *find* $\mathbf{u} \in \mathbf{V}(\mathbf{g})$ *and* $\tilde{p} \in L^2(\Omega)$ *such that*

$$\begin{array}{rcccccl} Re\, \tilde{t}^*(\mathbf{u};\, \mathbf{u},\, \mathbf{v}) & + & a(\mathbf{u},\mathbf{v}) & + & b(\mathbf{v},\tilde{p}) & = & \tilde{l}(\mathbf{v}),\ \forall \mathbf{v} \in \mathbf{V}(0), \\ & & b(\mathbf{u},q) & & & = & 0,\ \forall q \in L^2(\Omega) \end{array}$$

One can easily check that when $(\mathbf{u},\tilde{p})$ is a solution of $(\widetilde{FV})^*$, then $(\mathbf{u},p)$ is a solution of $(\widetilde{FV})$ with $p = \tilde{p} + Re|\mathbf{u}|/2$. The apparition of the kinetic energy term $Re|\mathbf{u}|/2$ in the modified pressure field $\tilde{p}$ is due to the introduction of the $\mathrm{div}(\mathbf{v})\,(\mathbf{u}.\mathbf{w})$ term in the trilinear form $\tilde{t}^*$.
At the discrete level, let us define for all $\mathbf{u}_h, \mathbf{v}_h, \mathbf{w}_h \in \mathbf{X}_h$:

$$\begin{aligned} \tilde{t}_h^*(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) \quad = \quad & -\int_\Omega \left((\mathbf{w}_h \otimes \mathbf{u}_h) : \nabla_h \mathbf{v}_h + \frac{1}{2}\mathrm{div}_h(\mathbf{v}_h)\,(\mathbf{u}_h.\mathbf{w}_h) \right) \mathrm{d}x \\ & + \frac{1}{2}\int_{\partial\Omega} (\mathbf{v}_h.\mathbf{n})\,(\mathbf{u}_h.\mathbf{w}_h)\,\mathrm{d}s \end{aligned}$$

Notice that $\tilde{t}_h^*$ is similar to $\tilde{t}^*$: the gradient and divergence has been replaced by their broken counterpart in the first term. As $\mathbf{X}_h \not\subset H^1(\Omega)^d$, the skew-symmetry property is not expected to be true at the discrete level. Then

$$\tilde{t}_h^*(\mathbf{u}_h;\, \mathbf{u}_h,\, \mathbf{u}_h) \quad = \quad -\int_\Omega \left((\mathbf{u}_h \otimes \mathbf{u}_h) : \nabla_h \mathbf{u}_h + \frac{1}{2}\mathrm{div}_h(\mathbf{u}_h)\,|\mathbf{u}_h|^2 \right) \mathrm{d}x + \frac{1}{2}\int_{\partial\Omega} (\mathbf{u}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s$$

Next, using (3.9) in each K, and then developing thanks to (3.5)-(3.6), we get

$$\begin{aligned} \tilde{t}_h^*(\mathbf{u}_h;\, \mathbf{u}_h,\, \mathbf{u}_h) \quad &= \quad \frac{1}{2}\int_{\partial\Omega} (\mathbf{u}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s - \frac{1}{2}\sum_{K\in\mathscr{T}_h}\int_{\partial K} (\mathbf{u}_h.\mathbf{n})\,|\mathbf{u}_h|^2\,\mathrm{d}s \\ &= \quad -\frac{1}{2}\sum_{S\in\mathscr{S}_h^{(i)}}\int_S [\![(\mathbf{u}_h.\mathbf{n})\,|\mathbf{u}_h|^2]\!]\,\mathrm{d}s \\ &= \quad -\frac{1}{2}\sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left((\{\!\{\mathbf{u}_h\}\!\}.\mathbf{n})\,[\![|\mathbf{u}_h|^2]\!] + ([\![\mathbf{u}_h]\!].\mathbf{n})\,\{\!\{|\mathbf{u}_h|^2\}\!\} \right) \mathrm{d}s \\ &= \quad -\sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left((\{\!\{\mathbf{u}_h\}\!\}.\mathbf{n})\,(\{\!\{\mathbf{u}_h\}\!\}.[\![\mathbf{u}_h]\!]) + \frac{1}{2}([\![\mathbf{u}_h]\!].\mathbf{n})\,\{\!\{|\mathbf{u}_h|^2\}\!\} \right) \mathrm{d}s \end{aligned}$$

The idea is to integrate this term in the definition of a discrete $\tilde{t}_h$. One of the possibilities is

$$\begin{aligned} \tilde{t}_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) \quad &= \quad \tilde{t}_h^*(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) + \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left((\{\!\{\mathbf{u}_h\}\!\}.\mathbf{n})\,(\{\!\{\mathbf{w}_h\}\!\}.[\![\mathbf{v}_h]\!]) + \frac{1}{2}\{\!\{\mathbf{u}_h.\mathbf{w}_h\}\!\}\,([\![\mathbf{v}_h]\!].\mathbf{n}) \right) \mathrm{d}s \\ &= \quad -\int_\Omega \left((\mathbf{w}_h \otimes \mathbf{u}_h) : \nabla_h \mathbf{v}_h + \frac{1}{2}\mathrm{div}_h(\mathbf{v}_h)\,(\mathbf{u}_h.\mathbf{w}_h) \right) \mathrm{d}x \\ &\quad + \frac{1}{2}\int_{\partial\Omega} (\mathbf{v}_h.\mathbf{n})\,(\mathbf{u}_h.\mathbf{w}_h)\,\mathrm{d}s \\ &\quad + \sum_{S\in\mathscr{S}_h^{(i)}}\int_S \left((\{\!\{\mathbf{u}_h\}\!\}.\mathbf{n})\,(\{\!\{\mathbf{w}_h\}\!\}.[\![\mathbf{v}_h]\!]) + \frac{1}{2}\{\!\{\mathbf{u}_h.\mathbf{w}_h\}\!\}\,([\![\mathbf{v}_h]\!].\mathbf{n}) \right) \mathrm{d}s \end{aligned} \tag{3.10}$$

This expression was proposed by [10, p. 21], eqn (73) (see also [11, p. 282]) folling and original idea introduced in [8].

Example file 3.7: `inertia_cks.icc`

```
form inertia (field w, trial u, test v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return
    integrate (- dot(trans(grad_h(v))*w,u) - 0.5*div_h(v)*dot(u,w), qopt)
  + integrate ("internal_sides",
            dot(average(u),normal())*dot(jump(v),average(w))
     + 0.5*dot(jump(v),normal())
            *(dot(average(u),average(w)) + 0.25*dot(jump(u),jump(w))), qopt)
  + integrate ("boundary", 0.5*dot(v,normal())*dot(u,w), qopt);
}
field inertia_fix_rhs (test v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return integrate("boundary", -dot(g(),normal())*dot(g(),v), qopt);
}
```

The discrete problem is

$(\widetilde{FV})_h$: *find* $\mathbf{u}_h \in \mathbf{X}_h$ *and* $\tilde{p} \in Q_h$ *such that*

$$\begin{array}{rcccccl} Re\,\tilde{t}_h(\mathbf{u}_h;\,\mathbf{u}_h,\,\mathbf{v}_h) & + & a_h(\mathbf{u}_h,\mathbf{v}_h) & + & b_h(\mathbf{v}_h,\tilde{p}_h) & = & \tilde{l}_h^*(\mathbf{v}_h),\ \forall \mathbf{v}_h \in \mathbf{X}_h, \\ & & b_h(\mathbf{u}_h,q_h) & - & c_h(p_h,q_h) & = & k_h(q),\ \forall q_h \in Q_h \end{array}$$

A simple test program is obtained by replacing in `navier_stokes_taylor_dg.cc` the include `inertia.icc` by `inertia_cks.icc`. The compilation and run are similar.

3.3.4 A Newton solver

The discrete problems $(FV)_h$ can be put in a compact form:

$$F(\mathbf{u}_h, p_h) = 0$$

where F is defined in variationnal form:

$$\langle F(\mathbf{u}_h,p_h),\,(\mathbf{v}_h,q_h)\rangle = \left(\begin{array}{ccccccc} Re\,t_h(\mathbf{u}_h;\,\mathbf{u}_h,\,\mathbf{v}_h) & + & a_h(\mathbf{u}_h,\mathbf{v}_h) & + & b_h(\mathbf{v}_h,p_h) & - & l_h^*(\mathbf{v}_h) \\ & & b_h(\mathbf{u}_h,q_h) & - & c_h(p_h,q_h) & - & k_h(q) \end{array}\right)$$

for all $(\mathbf{v}_h,q_h) \in \mathbf{X}_h \times Q_h$. Notices that, after some minor modifications in the definition of F, this method could also applies for the locally conservative formulation $(\widetilde{FV})_h$. The previous formulation is simply the variationnal expression of $F(\mathbf{u}_h,p_h) = 0$. The Newton method defines the sequence $\left(\mathbf{u}_h^{(k)}\right)_{k\geqslant 0}$ by reccurence as:

- $k = 0$: let $\mathbf{u}_h^{(0)} \in \mathbf{X}_h$ being known.
- $k \geqslant 0$: let $\mathbf{u}_h^{(k-1)} \in \mathbf{X}_h$ given. Find $\delta\mathbf{u}_h \in \mathbf{X}_h$ and $\delta p_h \in Q_h$ such that

  $$F'\left(\mathbf{u}_h^{(k-1)},\,p_h^{(k-1)}\right).(\delta\mathbf{u}_h,\delta p_h) = -F\left(\mathbf{u}_h^{(k-1)},p_h^{(k-1)}\right)$$

  and then defines

  $$\mathbf{u}_h^{(k)} = \mathbf{u}_h^{(k-1)} + \delta\mathbf{u}_h \quad \text{and} \quad p_h^{(k)} = p_h^{(k-1)} + \delta p_h$$

At each step $k \geqslant 0$, this algorithm involves a linear subproblem involving the jacobian F' that is definied by its variationnal form:

$$\begin{array}{rl} & \langle F'\left(\mathbf{u}_h^{(k-1)},\,p_h^{(k-1)}\right).(\delta\mathbf{u}_h,\delta p_h),\ (\mathbf{v}_h,q_h)\rangle \\ = & \left(\begin{array}{ccccc} Re\ (t_h(\delta\mathbf{u}_h;\,\mathbf{u}_h,\,\mathbf{v}_h) + t_h(\mathbf{u}_h;\,\delta\mathbf{u}_h,\,\mathbf{v}_h)) & + & a_h(\delta\mathbf{u}_h,\mathbf{v}_h) & + & b_h(\mathbf{v}_h,\delta p_h) \\ & & b_h(\delta\mathbf{u}_h,q_h) & - & c_h(\delta p_h,q_h) \end{array}\right) \end{array}$$

Example file 3.8: navier_stokes_taylor_newton_dg.cc

```
#include "rheolef.h"
using namespace rheolef;
using namespace std;
#include "taylor.icc"
#include "stokes_dirichlet_dg.icc"
#include "inertia.icc"
#include "navier_stokes_dg.h"
int main(int argc, char**argv) {
  environment rheolef (argc, argv);
  Float eps = numeric_limits<Float>::epsilon();
  geo omega (argv[1]);
  string approx   = (argc > 2) ?      argv[2]  : "P1d";
  Float  Re       = (argc > 3) ? atof(argv[3]) : 100;
  Float  tol      = (argc > 4) ? atof(argv[4]) : eps;
  size_t max_iter = (argc > 5) ? atoi(argv[5]) : 100;
  string restart  = (argc > 6) ?      argv[6]  : "";
  navier_stokes_dg F (Re, omega, approx);
  navier_stokes_dg::value_type xh = F.initial (restart);
  int status = damped_newton (F, xh, tol, max_iter, &derr);
  dout << catchmark("Re") << Re << endl
       << catchmark("u") << xh[0]
       << catchmark("p") << xh[1];
  return status;
}
```

Comments

The implementation of the Newton method follows the generic approach introduced in volume 1, section 8.3, page 132. For that purpose we define a class navier_stokes_dg.

Example file 3.9: navier_stokes_dg.h

```
struct navier_stokes_dg {
  typedef valarray<field> value_type;
  typedef Float           float_type;
  navier_stokes_dg (Float Re, const geo& omega, string approx);
  value_type initial (string restart) const;
  value_type residue     (const value_type& uh) const;
  void update_derivative (const value_type& uh) const;
  value_type derivative_solve      (const value_type& mrh) const;
  value_type derivative_trans_mult (const value_type& mrh) const;
  Float space_norm       (const value_type& uh) const;
  Float dual_space_norm  (const value_type& mrh) const;
  Float Re;
  space Xh, Qh;
  quadrature_option_type qopt;
  form  a0, b, c, mu, mp;
  field lh0, lh, kh;
  solver smu, smp;
  mutable form a1;
  mutable solver_abtb stokes1;
};
#include "navier_stokes_dg1.icc"
#include "navier_stokes_dg2.icc"
```

The member functions of the class are defined in two separate files.

Example file 3.10: navier_stokes_dg1.icc

```
navier_stokes_dg::navier_stokes_dg (
 Float Re1, const geo& omega, string approx)
 : Re(Re1), Xh(), Qh(), qopt(), a0(), b(), c(), mu(), mp(), lh0(), lh(), kh(),
   smu(), smp(), a1(), stokes1()
{
  Xh = space (omega, approx, "vector");
  Qh = space (omega, approx);
  qopt.set_family(quadrature_option_type::gauss);
  qopt.set_order(2*Xh.degree()+1);
  stokes_dirichlet_dg (Xh, Qh, a0, b, c, mp, lh0, kh, qopt);
  trial u (Xh); test v (Xh);
  lh = lh0 + Re*inertia_fix_rhs (v, qopt);
  mu = integrate (dot(u,v), qopt);
  smu = solver(mu.uu());
  smp = solver(mp.uu());
}
navier_stokes_dg::value_type
navier_stokes_dg::initial (string restart) const {
  value_type xh(2);
  xh[0] = field (Xh, 0);
  xh[1] = field (Qh, 0);
  Float Re0 = 0;
  if (restart == "") {
    solver_abtb stokes0 (a0.uu(), b.uu(), c.uu(), mp.uu());
    stokes0.solve (lh0.u(), kh.u(), xh[0].set_u(), xh[1].set_u());
  } else {
    idiststream in (restart);
    in >> catchmark("Re") >> Re0
       >> catchmark("u")  >> xh[0]
       >> catchmark("p")  >> xh[1];
    check_macro (xh[1].get_space() == Qh, "unexpected " << xh[0].get_space().stamp()
        << " approximation in file \"" << restart << "\" (" << Xh.stamp() << " expected)");
  }
  derr << "# continuation: from Re=" << Re0 << " to " << Re << endl;
  return xh;
}
navier_stokes_dg::value_type
navier_stokes_dg::residue (const value_type& xh) const {
  trial u (Xh); test v (Xh);
  form a = a0 + Re*inertia(xh[0], u, v, qopt);
  value_type mrh(2);
  mrh[0] = a*xh[0] + b.trans_mult(xh[1]) - lh;
  mrh[1] = b*xh[0] -             c*xh[1]  - kh;
  return mrh;
}
void navier_stokes_dg::update_derivative (const value_type& xh) const {
  trial u (Xh); test v (Xh);
  a1 = a0 + Re*(inertia(xh[0], u, v, qopt) + inertia(u, xh[0], v, qopt));
  stokes1 = solver_abtb (a1.uu(), b.uu(), c.uu(), mp.uu());
}
navier_stokes_dg::value_type
navier_stokes_dg::derivative_solve (const value_type& mrh) const {
  value_type delta_xh(2);
  delta_xh[0] = field (Xh, 0);
  delta_xh[1] = field (Qh, 0);
  stokes1.solve (mrh[0].u(), mrh[1].u(),
                 delta_xh[0].set_u(), delta_xh[1].set_u());
  return delta_xh;
}
navier_stokes_dg::value_type
navier_stokes_dg::derivative_trans_mult (const value_type& mrh) const {
  value_type rh(2);
  rh[0] = field (Xh);
  rh[1] = field (Qh);
  rh[0].set_u() = smu.solve(mrh[0].u());
  rh[1].set_u() = smp.solve(mrh[1].u());
  value_type mgh(2);
  mgh[0] = a1.trans_mult(rh[0]) + b.trans_mult(rh[1]);
  mgh[1] =              b*rh[0]  -             c*rh[1];
  return mgh;
}
```

Example file 3.11: navier_stokes_dg2.icc

```
Float navier_stokes_dg::space_norm (const value_type& xh) const {
  return sqrt (mu(xh[0],xh[0]) + mp(xh[1],xh[1]));
}
Float navier_stokes_dg::dual_space_norm (const value_type& mrh) const {
  value_type rh(2);
  rh[0] = field (Xh,0);
  rh[1] = field (Qh,0);
  rh[0].set_u() = smu.solve(mrh[0].u());
  rh[1].set_u() = smp.solve(mrh[1].u());
  return sqrt (dual(rh[0],mrh[0]) + dual(rh[1],mrh[1]));
}
```

```
make navier_stokes_taylor_newton_dg navier_stokes_taylor_error_dg
./navier_stokes_taylor_newton_dg square P2d 1000 | ./navier_stokes_taylor_error_dg
```

3.3.5 Application to the driven cavity benchmark

Example file 3.12: cavity_dg.icc

```
struct g {
  point operator() (const point& x) const {
    return point((abs(1-x[1]) < 1e-7) ? 1 : 0, 0, 0); }
};
struct f {
  point operator() (const point& x) const { return point(0,0,0); }
};
```

The program navier_stokes_cavity_newton_dg.cc is obtained by replacing in navier_stokes_taylor_newton_dg.cc the include taylor.icc by cavity_dg.icc that defines the boundary conditions. The compilation and run are similar.

```
make navier_stokes_cavity_newton_dg streamf_cavity
./navier_stokes_cavity_newton_dg  square P1d 500 > square.field
field -proj square.field -field | ./streamf_cavity | \
                   field -bw -n-iso-negative 10 -
```

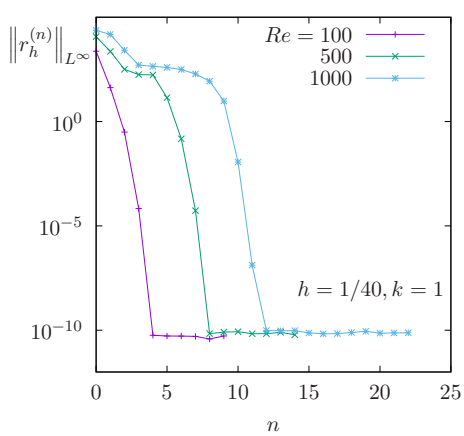


Figure 3.1: The discontinuous Galerkin method for the Navier-Stokes problem on the driven cavity benchmark when $k = 1$ and $d = 2$: convergence of the damped Newton algorithm.

3.3.6 Upwinding

The skew symmetry property is generalized to the requirement that t_h be non-dissipative (see [11, p. 282], eqn (6.68)):

$$t_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{u}_h) \geqslant 0,\ \forall \mathbf{w}_h,\, \mathbf{u}_h \in \mathbf{X}_h$$

A way to satisfy this property is to add an *upwinding* term in t_h:

$$\breve{t}_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) := t_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) + s_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h)$$

with

$$s_h(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) = \frac{1}{2} \sum_{S \in \mathscr{S}_h^{(i)}} \int_S |\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n}|\ (\llbracket \mathbf{u}_h \rrbracket.\llbracket \mathbf{v}_h \rrbracket)\, \mathrm{d}s$$

We aim at using a Newton method. We replace t_h by its extension $\breve{t}_h$ containing the upwind terms in the definition of F, and then we compute its jacobian F'. As the absolute value is not differentiable, the functions s_h, $\breve{t}_h$ and then F are also not differentiable with respect to $\mathbf{w}_h$. Nevertheless, the absolute value is convex and we can use some concets of the sudifferential calculus. Let us introduce the multi-valued sign function:

$$\mathrm{sgn}(x) = \begin{cases} \{1\} & \text{when } x > 0 \\ [-1, 1] & \text{when } x = 0 \\ \{-1\} & \text{when } x < 0 \end{cases}$$

Then, the subdifferential of the absolute value function is $\mathrm{sgn}(x)$ and for all $\delta\mathbf{w}_h, \mathbf{w}_h,\, \mathbf{u}_h, \mathbf{v}_h \in \mathbf{X}_h$, we define a generalization of the partial derivative as

$$\frac{\partial s_h}{\partial \mathbf{w}_h}(\mathbf{w}_h;\, \mathbf{u}_h,\, \mathbf{v}_h).(\delta\mathbf{w}_h) = \frac{1}{2} \sum_{S \in \mathscr{S}_h^{(i)}} \int_S \mathrm{sgn}(\{\!\{\mathbf{w}_h\}\!\}.\mathbf{n})\ (\{\!\{\delta\mathbf{w}_h\}\!\}.\mathbf{n})\ (\llbracket \mathbf{u}_h \rrbracket.\llbracket \mathbf{v}_h \rrbracket)\, \mathrm{d}s$$

Example file 3.13: `inertia_upw.icc`

```
#include "sgn.icc"
form inertia_upw (field w, trial u, test v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return integrate ("internal_sides",
     0.5*abs(dot(average(w),normal()))*dot(jump(u),jump(v)));
}
form d_inertia_upw (field w, trial dw, field u, test v,
  quadrature_option_type qopt = quadrature_option_type())
{
  return integrate ("internal_sides",
     0.5*compose (sgn, dot(average(w),normal()))
         *dot(average(dw),normal())*dot(jump(u),jump(v)));
}
```

A multi-valued jacobian F' is then defined:

$$\begin{aligned}
& \langle F'\left(\mathbf{u}_h^{(k-1)},\, p_h^{(k-1)}\right).(\delta\mathbf{u}_h, \delta p_h),\ (\mathbf{v}_h, q_h)\rangle \\
= \quad & Re \begin{pmatrix} t_h(\delta\mathbf{u}_h;\, \mathbf{u}_h,\, \mathbf{v}_h) + t_h(\mathbf{u}_h;\, \delta\mathbf{u}_h,\, \mathbf{v}_h) + \dfrac{\partial s_h}{\partial \mathbf{w}_h}(\mathbf{u}_h;\, \mathbf{u}_h,\, \mathbf{v}_h).(\delta\mathbf{u}_h) + s_h(\mathbf{u}_h;\, \delta\mathbf{u}_h,\, \mathbf{v}_h) \\ \\ 0 \end{pmatrix} \\
& + \begin{pmatrix} a_h(\delta\mathbf{u}_h, \mathbf{v}_h) & + & b_h(\mathbf{v}_h, \delta p_h) \\ \\ b_h(\delta\mathbf{u}_h, q_h) & - & c_h(\delta p_h, q_h) \end{pmatrix}
\end{aligned}$$

We are abble to extend the Newton method to the F function that allows a multi-valued subdifferential F'. During iterations, we can choose any of the available directions in the subdifferential. One the possibilities is then to replace the multi-valued sign function by a single-value one:

$$\widetilde{\mathrm{sgn}}(x) = \begin{cases} 1 & \text{when } x \geqslant 0 \\ -1 & \text{when } x < 0 \end{cases}$$

Example file 3.14: `sgn.icc`

```
Float sgn (Float x) { return (x >= 0) ? 1 : -1; }
```

Example file 3.15: `navier_stokes_upw_dg.h`

```
#include "navier_stokes_dg.h"
struct navier_stokes_upw_dg: navier_stokes_dg {
  typedef valarray<field> value_type;
  typedef Float           float_type;
  navier_stokes_upw_dg (Float Re, const geo& omega, string approx);
  value_type residue     (const value_type& uh) const;
  void update_derivative (const value_type& uh) const;
};
#include "navier_stokes_upw_dg.icc"
```

Example file 3.16: `navier_stokes_upw_dg.icc`

```
#include "inertia_upw.icc"
navier_stokes_upw_dg::navier_stokes_upw_dg (
 Float Re1, const geo& omega, string approx)
 : navier_stokes_dg (Re1, omega, approx) {}

navier_stokes_upw_dg::value_type
navier_stokes_upw_dg::residue (const value_type& xh) const {
  trial u (Xh); test v (Xh);
  form a = a0 + Re*(  inertia     (xh[0], u, v, qopt)
                    + inertia_upw (xh[0], u, v, qopt));
  value_type mrh(2);
  mrh[0] = a*xh[0] + b.trans_mult(xh[1]) - lh;
  mrh[1] = b*xh[0] -             c*xh[1]  - kh;
  return mrh;
}
void navier_stokes_upw_dg::update_derivative (const value_type& xh) const {
  trial du (Xh); test v (Xh);
  a1 = a0 + Re*(      inertia          (xh[0], du,    v, qopt)
                  + inertia_upw        (xh[0], du,    v, qopt)
                  +    inertia         (du,    xh[0], v, qopt)
                + d_inertia_upw (xh[0], du,    xh[0], v, qopt));
  stokes1 = solver_abtb (a1.uu(), b.uu(), c.uu(), mp.uu());
}
```

The program `navier_stokes_cavity_newton_upw_dg.cc` is obtained by replacing in `navier_stokes_taylor_newton_dg.cc` the string `navier_stokes_dg` by `navier_stokes_upw_dg` (two occurences: in the includes and then in the definition of `F`). Also replace the include `taylor.icc` by `cavity_dg.icc` that defines the boundary conditions. The compilation and run are similar.

```
make navier_stokes_cavity_newton_upw_dg stream_cavity
mkgeo_grid -t 80 > square.geo
./navier_stokes_cavity_newton_upw_dg  square P1d 500 1e-15 100 > square-500.field
field -proj square-500.field -field | ./streamf_cavity | \
                  field -bw -n-iso 30 -n-iso-negative 20 -
```

Computations for higher Renolds numbers are performed by continuation, starting from a previous computation at lower Re:

```
./navier_stokes_cavity_newton_upw_dg  square P1d 1000 1e-15 100 square-500.field  > square-1000.fi
```

```
./navier_stokes_cavity_newton_upw_dg  square P1d 1500 1e-15 100 square-1000.field > square-1500.fi
```

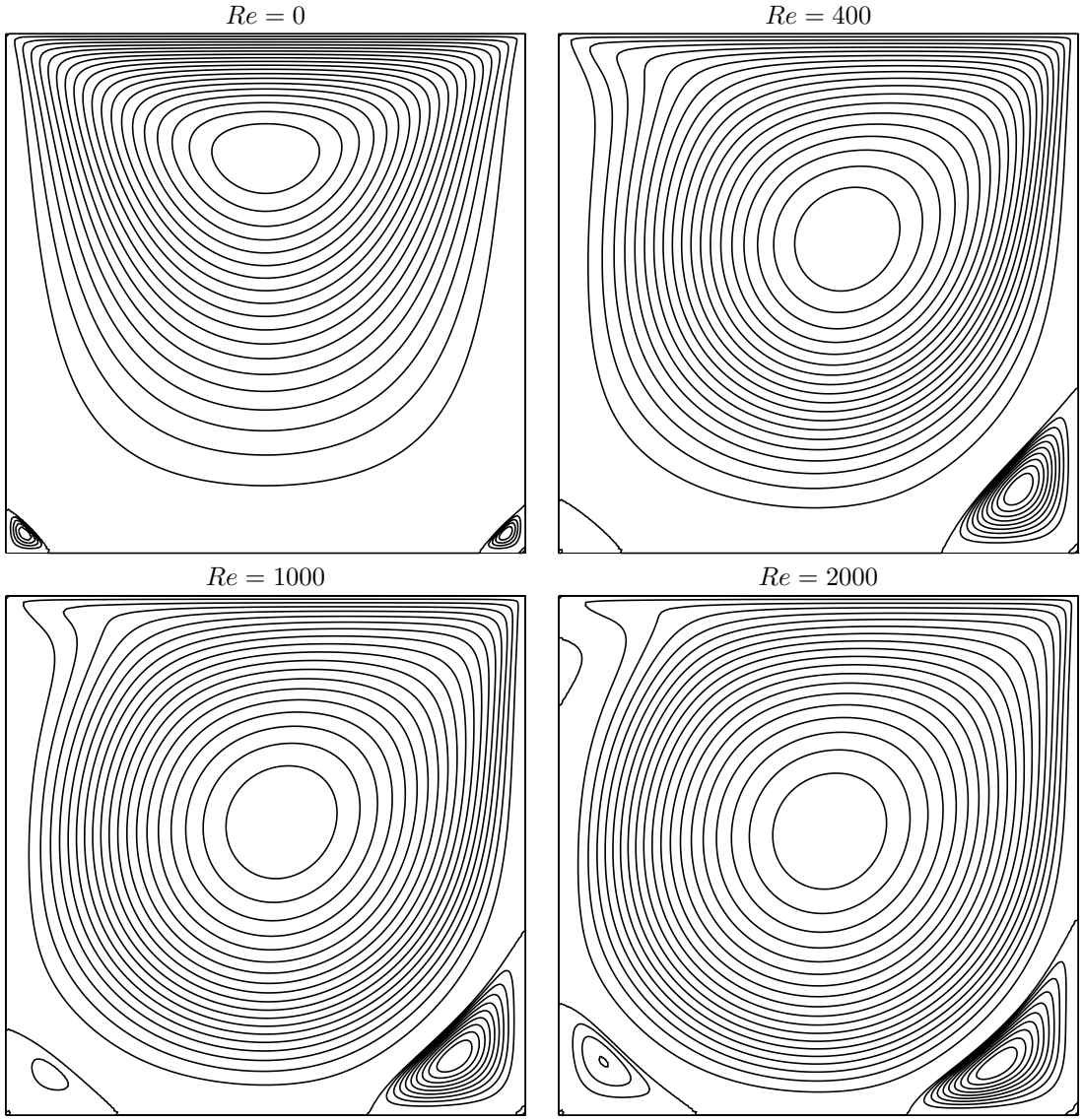


Figure 3.2: The discontinuous Galerkin method for the Navier-Stokes problem on the driven cavity benchmark when $k = 1$ (80×80 grid): stream function isovalues for various Re.

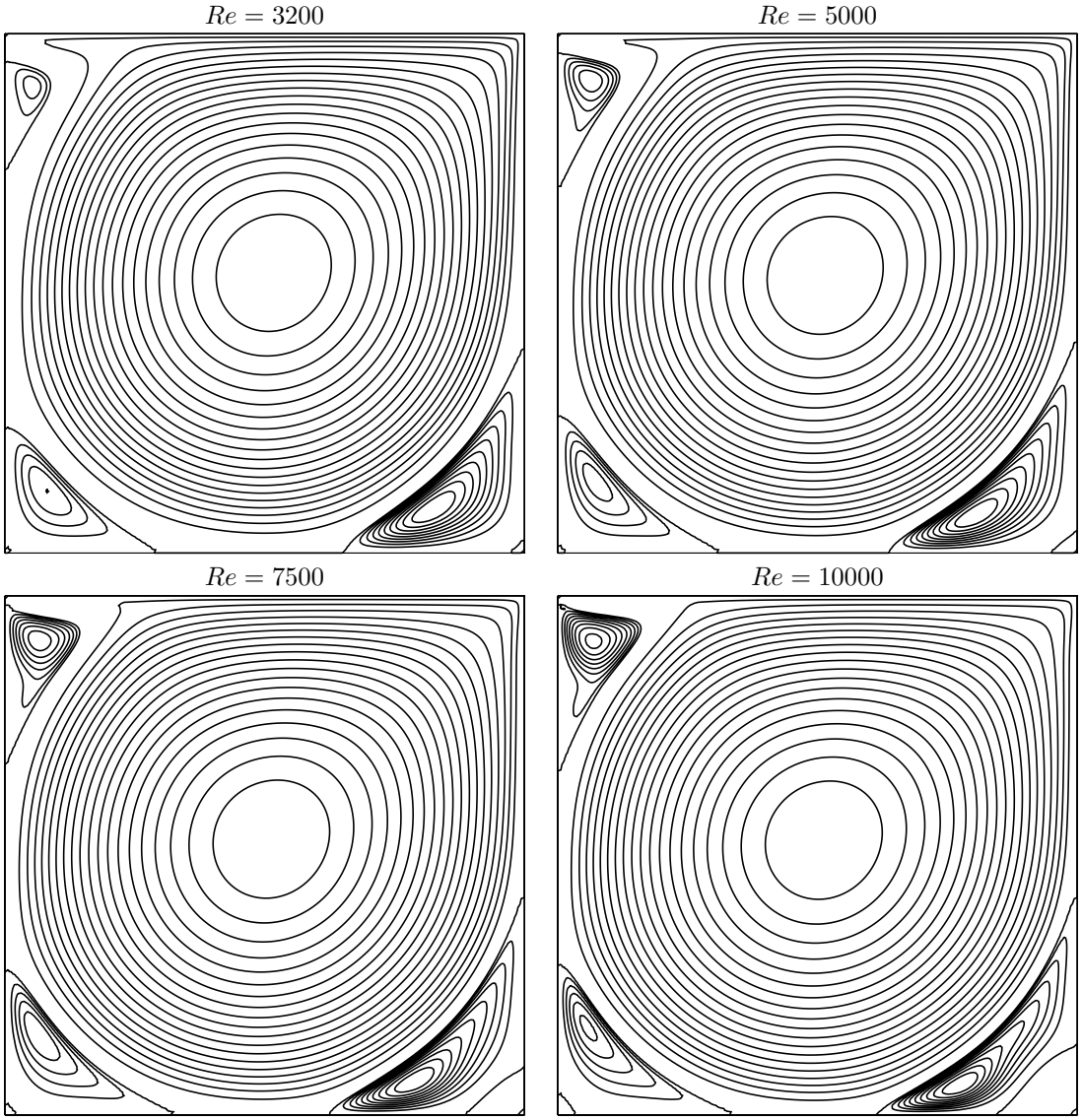


Figure 3.3: The discontinuous Galerkin method for the Navier-Stokes problem on the driven cavity benchmark when $k = 1$ (80×80 grid): stream function isovalues for various Re (cont.).

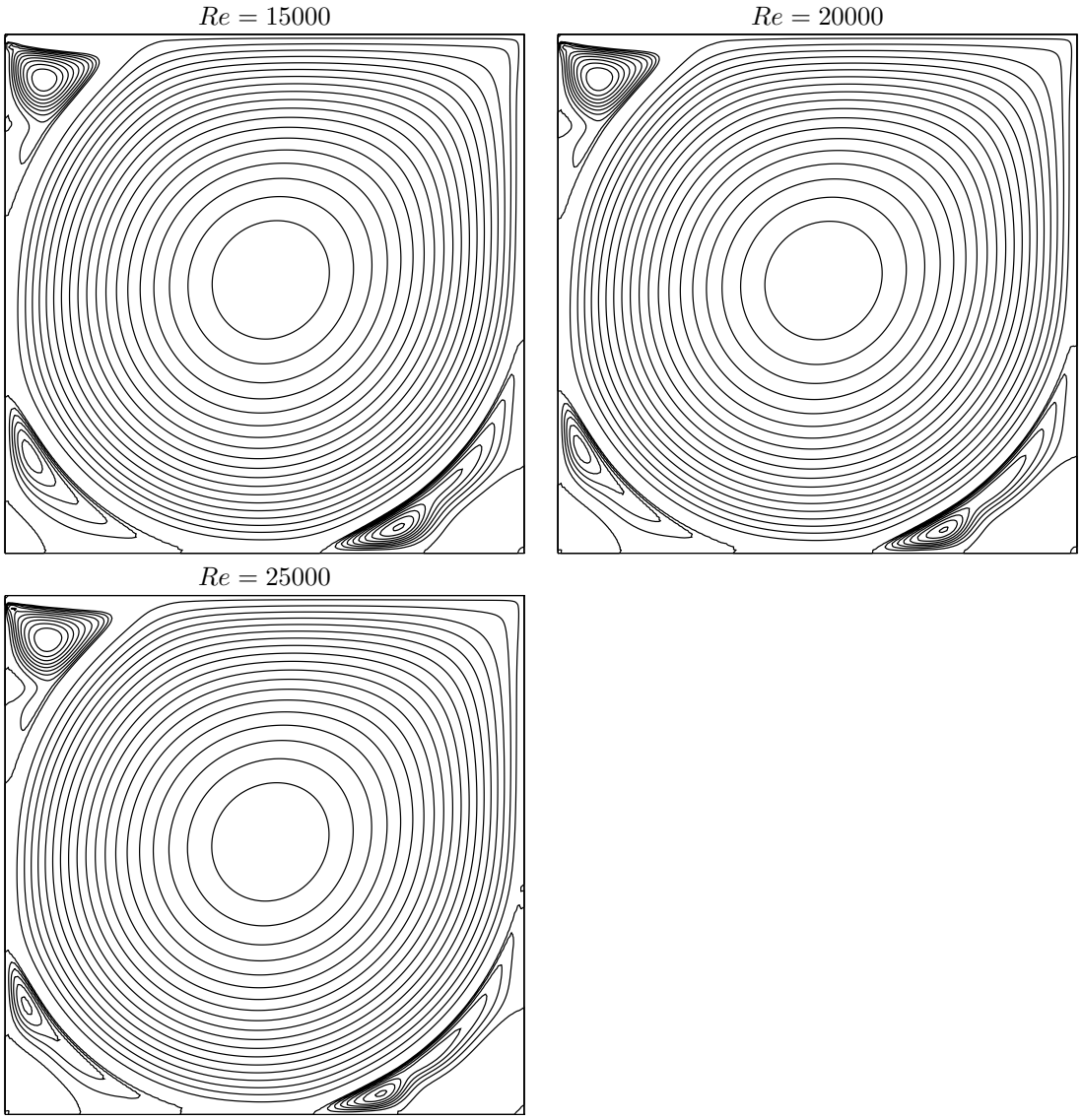


Figure 3.4: The discontinuous Galerkin method for the Navier-Stokes problem on the driven cavity benchmark when $k = 1$ (80×80 grid): stream function isovalues for various Re (cont.).

Part III

Technical appendices

Appendix A

GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

*

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary

Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single

copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

*

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliography

[1] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25(2):151–167, 1997. 27, 28

[2] M. P. Calvo, J. de Frutos, and J. Novo. Linearly implicit Runge-Kutta methods for advection-reaction-diffusion equations. *Appl. Numer. Math.*, 37(4):535–549, 2001. 27, 28

[3] G. F. Carey and B. Jianng. Least-squares finite elements for first-order hyperbolic systems. *Int. J. Numer. Meth. Eng.*, 26(1):81–93, 1988. 14

[4] P. Castillo. Performance of discontinuous Galerkin methods for elliptic PDEs. *SIAM J. Sci. Comput.*, 24(2):524–547, 2002. 22

[5] B. Cockburn. *An introduction to the discontinuous Galerkin method for convection-dominated problems*, chapter 2, pages 151–268. Springer, 1998. 12

[6] B. Cockburn, B. Dong, J. Guzmán, and J. Qian. Optimal convergence of the original DG method on special meshes for variable transport velocity. *SIAM J. Numer. Anal.*, 48(1):133–146, 2010. 9

[7] B. Cockburn, S. Hou, and C.-W. Shu. The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV. the multidimensional case. *Math. Comput.*, 54(190):545–581, 1990. 14

[8] B. Cockburn, G. Kanschat, and D. Schötzau. A locally conservative LDG method for the incompressible Navier-Stokes equations. *Math. Comput.*, 74(251):1067–1095, 2005. 43

[9] B. Cockburn, G. Kanschat, D. Schötzau, and C. Schwab. Local discontinuous Galerkin methods for the Stokes system. *SIAM J. Numer. Anal.*, 40(1):319–343, 2002. 35

[10] D. A. di Pietro and A. Ern. Discrete functional analysis tools for discontinuous Galerkin methods with application to the incompressible Navier-Stokes equations. *Math. Comp.*, 79:1303–1330, 2010. 35, 39, 43

[11] D. A. di Pietro and A. Ern. *Mathematical aspects of discontinuous Galerkin methods*. Springer, 2012. 7, 8, 10, 11, 21, 23, 24, 27, 36, 39, 43, 48

[12] Y. Epshteyn and B. Rivière. Estimation of penalty parameters for symmetric interior penalty Galerkin methods. *J. Comput. Appl. Math.*, 206(2):843–872, 2007. 21

[13] S. Gottlieb and C.-W. Shu. Total variation diminishing Runge-Kutta schemes. *Math. Comput.*, 67(221):73–85, 1998. 11, 12

[14] S. Gottlieb, Chi-W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM review*, 43(1):89–112, 2001. 11, 12

[15] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *J. Comput. Phys.*, 71(2):231–303, 1987. 15

[16] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods. Algorithms, analysis and applications*. Springer, 2008. 7

[17] C. Johnson and J. Pitkäranta. An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation. *Math. Comp.*, 46(173):1–26, 1986. 9

[18] T. E. Peterson. A note on the convergence of the discontinuous Galerkin method for a scalar hyperbolic equation. *SIAM J. Numer. Anal.*, 28(1):133–140, 1991. 9

[19] G. R. Richter. An optimal-order error estimate for the discontinuous galerkin method. *Math. Comput.*, 50(181):75–88, 1988. 9

[20] K. Shahbazi. An explicit expression for the penalty parameter of the interior penalty method. *J. Comput. Phys.*, 205(2):401–407, 2005. 22

[21] C.-W. Shu. TVB boundary treatment for numerical solutions of conservation laws. *Math. Comput.*, 49(179):123–134, 1987. 13

[22] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J. Comput. Phys.*, 77(2):439–471, 1988. 11, 12

[23] G. I. Taylor. On the decay of vortices in a viscous fluid. *Philos. Mag.*, 46:671–674, 1923. 34

[24] H. Wang, C.-W. Shu, and Q. Zhang. Stability analysis and error estimates of local discontinuous Galerkin methods with implicit–explicit time-marching for nonlinear convection–diffusion problems. *Appl. Math. Comput.*, 2015. 27

[25] H. Wang, C.-W. Shu, and Q. Zhang. Stability and error estimates of local discontinuous Galerkin methods with implicit-explicit time-marching for advection-diffusion problems. *SIAM J. Numer. Anal.*, 53(1):206–227, 2015. 27, 28, 30

[26] X. Zhong and C.-W. Shu. A simple weighted essentially nonoscillatory limiter for Runge-Kutta discontinuous galerkin methods. *J. Comput. Phys.*, 232(1):397–415, 2013. 18

List of example files

burgers.icc, 14
burgers_dg.cc, 16
burgers_diffusion_dg.cc, 28
burgers_diffusion_exact.icc, 26
burgers_diffusion_operators.icc, 29
burgers_flux_godunov.icc, 14
cavity_dg.icc, 47
dirichlet_dg.cc, 22
elasticity_taylor_dg.cc, 34
harten.icc, 15
harten_show.cc, 15
inertia.icc, 39
inertia_cks.icc, 43
inertia_upw.icc, 48
navier_stokes_dg.h, 45
navier_stokes_dg1.icc, 45
navier_stokes_dg2.icc, 47
navier_stokes_taylor_dg.cc, 40
navier_stokes_taylor_newton_dg.cc, 45
navier_stokes_upw_dg.h, 49
navier_stokes_upw_dg.icc, 49
neumann_dg.cc, 24
sgn.icc, 49
stokes_dirichlet_dg.icc, 36
stokes_taylor_dg.cc, 36
taylor.icc, 34
transport_dg.cc, 9
cosinusprod_error_dg.cc, 23
elasticity_taylor_error_dg.cc, 34
harten0.icc, 15
navier_stokes_cavity_newton_dg.cc, 47
navier_stokes_cavity_newton_upw_dg.cc, 49
navier_stokes_taylor_error_dg.cc, 41
runge_kutta_ssp.icc, 12
stokes_taylor_error_dg.cc, 37
taylor.icc, 37, 41

Index

approximation
- P0, 8
- P1, 37
- P2, 37
- discontinuous, 7

benchmark
- driven cavity flow, 35, 37
- embankment, 33

boundary condition
- Dirichlet, 21, 23, 35, 37
  - weakly imposed, 7, 21

broken Sobolev space $H^1(\mathcal{T}_h)$, 21, 26

convergence
- error
  - versus mesh, 9, 23

form
- $[\![u]\!]\{\!\{\nabla_h v.\mathbf{n}\}\!\}$, 21, 24, 27
- $[\![u]\!]\{\!\{v\}\!\}$, 8
- $[\![u]\!][\![v]\!]$, 8, 21, 24, 27

internal sides of a mesh, 8

method
- Euler explicit scheme, 12
- Runge-Kutta scheme, 11, 27
- upwind scheme, 48

operator
- `average`, accross sides, 8, 21
- `jump`, accross sides, 8, 21

penalty parameter, 22

problem
- Navier-Stokes, 37
- Poisson, 21, 23
- Stokes, 35, 37
- elasticity, 33
- nonlinear, 37

upstream boundary, 7

upwinding, 8