Exercises for Tutorial09

Generating Local Operators

In tutorial09 we learned how to use the `dune-codegen` module to generate local operators. In this exercise we will use this to generate some more complicated discretizations.

The build directory of this exercise is

```
dune-course/release-build/dune-pdelab-tutorials/tutorial09/
   ↪ exercise/task
```

The corresponding source directory can be found under

```
dune-course/dune/dune-pdelab-tutorials/tutorial09/exercise/
   ↪ task
```

or by following the symlink in the build directory

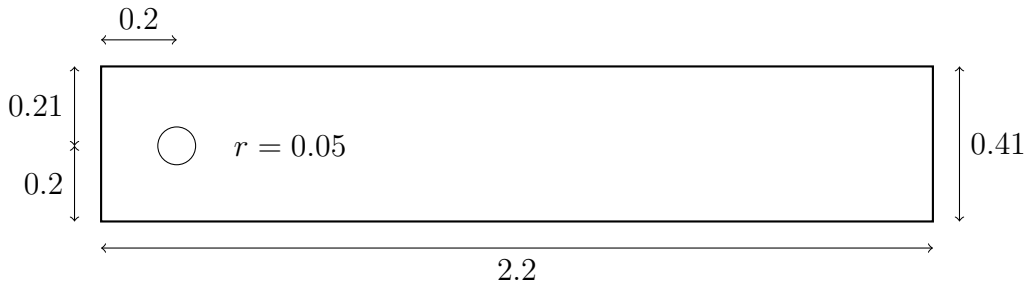
```
dune-course/release-build/dune-pdelab-tutorials/tutorial09/
   ↪ exercise/task/src_dir
```

Exercise 1 Navier Stokes

In this exercise we will implement the Navier Stokes equations for a flow around a cylinder[1] on the two dimenisional domain Ω shown below and the time interval Σ.

$$\begin{aligned} \rho\partial_t\vec{u} - \nu\Delta\vec{u} + \rho(\nabla\vec{u})\vec{u} + \nabla p = 0 \qquad & \text{in } \Omega \\ \nabla\cdot\vec{u} = 0 \qquad & \text{in } \Omega \end{aligned} \tag{1}$$

Here $\vec{u} : \Omega \times \Sigma \to \mathbb{R}^2$ is the unknown velocity field and $p : \Omega \times \Sigma \to \mathbb{R}$ is the unknown pressure. We assume that the kinematic viscosity $\nu = 0.001$ and the fluid density $\rho = 1$ are constant. The domain Ω is a rectangle with a circular hole.



[1] See http://www.featflow.de/en/benchmarks/cfdbenchmarking/flow/dfg_benchmark2_re100.html for a more detailed benchmark description

As boundary conditions we use

$$\begin{aligned} \vec{u} &= 0 && \text{on } \partial\Omega \cap (0, 2.2) \times [0, 0.41] \\ \vec{u} &= \begin{pmatrix} 1.5 \cdot 4 \cdot x_1 \frac{0.41 - x_1}{0.41^2} \cdot s(t) \\ 0 \end{pmatrix} && \text{on } 0 \times [0, 0.41] \\ \nu \nabla \vec{u} \vec{n} - p\vec{n} &= 0 && \text{on } 2.2 \times [0, 0.41] \end{aligned}$$

with the function

$$s(t) = \begin{cases} \sin\left(\frac{\pi t}{8}\right) & \text{if } t < 4 \\ 1 & \text{else} \end{cases}$$

The initial conditions are simply

$$\begin{aligned} \vec{u}|_{t=0} &= 0 \\ p|_{t=0} &= 0. \end{aligned}$$

A semi-discretization in space of this equation is: Find $(\vec{u}_h, p_h) \in U_h \times Q_h$ with

$$\rho(\partial_t \vec{u}_h, \vec{v}_h)_\Omega + r_h(\vec{u}_h, p_h, \vec{v}_h, q_h) = 0 \qquad \forall (\vec{v}_h, q_h) \in V_h \times Q_h$$

for appropriate function spaces U_h, V_h, Q_h and residual

$$r_h(\vec{u}, p, \vec{v}, q) = \nu(\nabla \vec{u}, \nabla \vec{v})_{0,\Omega} - (p, \nabla \cdot \vec{v})_{0,\Omega} - (q, \nabla \cdot \vec{u})_{0,\Omega} + \rho((\nabla \vec{u})\vec{u}, \vec{v})_{0,\Omega} \qquad (2)$$

Go to the source directory of this exercise. There you will find the files `navier_stokes.ufl` and `navier_stokes.ini`. Open the UFL file and implement the correct residual for the spatial discretization and the correct boundary conditions. For generating the C++ file and compiling go to the build directory and type

```
make navier_stokes
```

Help:

- UFL has a conditional:

$$\texttt{conditional(cond, A, B)} = \begin{cases} \texttt{A} & \texttt{cond is True} \\ \texttt{B} & \texttt{cond is False} \end{cases}$$

- UFL has `grad(.)` and `div(.)`
- `inner(.,.)*dx` will do the right thing for all scalar products of equation (2). Just make sure that the dimensions of the two arguments match.
- UFL has a method `sin(.)`.

Exercise 2 Nonlinear Poisson with Discontinuous Galerkin Method

In this exercise we will solve the nonlinear Poisson equation

$$\begin{aligned} -\Delta u + q(u) &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega \end{aligned} \qquad (3)$$

with the nonlinear function $q(u) = \eta u^2$ and the parameters functions

$$\begin{aligned} g(x) &= \|x\|_2^2 \\ f(x) &= -2d + \eta\, g(x)^2 \end{aligned}$$

using the discontinuous Galerkin (DG) method. In our case the dimension d is two and the parameter η describes the strength of the nonlinearity. You can for example set $\eta = 2$. We will focus on the implementation of this method in UFL to show the power of this approach without going into details about the numerical method.[2]

In contrast to continuous Galerkin methods DG methods use piecewise polynomial basis functions on the grid and allow for discontinuities along faces. In order to get a discritzation that still approximates the solution of our PDE penalty terms along the faces are introduced. Dirichlet boundary conditions are not build into the ansatz space but enforced in a weak way like it was done in exercise01 using Nitsche boundary condition.

A[3] DG discretization of problem (3) reads the following: Find u_h in U_h with

$$r_h(u_h, v_h) = 0 \qquad \forall v_h \in V_h$$

with the residual

$$\begin{aligned} r_h(u, v) = &\sum_{T \in \mathcal{T}_h} \int_T \nabla u \cdot \nabla v + q(u)v - fv \; dx \\ &- \sum_{F \in \mathcal{F}_h} \int_F (\{\nabla u\}, \vec{n})[\![v]\!] + [\![u]\!](\{\nabla v\}, \vec{n}) - \gamma_F [\![u]\!][\![v]\!] \; ds \\ &- \sum_{F \in \mathcal{B}_h} \int_F (\nabla u, \vec{n})v + (u - g)(\nabla v, \vec{n}) - \gamma_F (u - g)v \; ds \end{aligned} \tag{4}$$

We need to explain some notation: [4]

- $\mathcal{T}_h$: Set of mesh elements. Use `...*dx` in the UFL file for volume integrals. You don't need to care about the sum in front of the integral. UFL describes only the local integrals and will do the right thing.
- $\mathcal{B}_h$: Set of boundary faces. Use `...*ds` in the UFL file for integrals over boundary faces.
- $\mathcal{F}_h$: Set of inner faces. Use `...*dS` in the UFL file for integrals over inner faces.
- $\vec{n}$: Unit outer normal vector pointing from the inner cell to the outer cell. In the UFL file use `n = FacetNormal(cell)('+')`

[2] For further insight into DG methods see e.g. Di Pietro, Daniele Antonio and Ern, Alexandre: Mathematical aspects of discontinuous Galerkin methods

[3] There are different DG discretizations. We use symmetric interior penalty here.

[4] For faces we sometimes refer to the inner or the outer cell. For a given face it doesn't matter which cell is the inner and which is the outer as long as it is always treated the same for this face. This is not important for this exercise.

- $\{.\}$: Average of the values at the inside cell and the outside cell. Only makes sense at faces. In the UFL file use `avg(.)`.
- $[\![.]\!]$: Value at the inside cell minus value at the outside cell. Only makes sense at faces. In the UFL file use `jump(.)`.
- γ_F: Penalty parameter of the DG scheme. For this exercise we just choose $\gamma_F = 100$. A good choice of γ_F depends on the dimension, degree of your discretization and geometry informations. See the book mentioned above for further detail.

After reading all this text we can finally start doing some work:

1. Go to the source directory of this exercise. There you can find the files `nonlinear_poisson_dg.ufl` and `nonlinear_poisson_dg.mini` and a target is defined in the `CMakeLists.txt` file.
2. You need to implement the residual (4) in the UFL file.