Exercises for Tutorial03

Instationary Parabolic Equations

Exercise 1 Getting to Know the Code

The code of `tutorial03` solves the problem

$$\partial_t u - \Delta u + q(u) = f \qquad \text{in } \Omega \times \Sigma = (0,1)^d \times (t_0, T], \tag{1}$$
$$u(x,t) = g(x,t) \qquad \text{on } \Gamma_D \subseteq \partial\Omega, \tag{2}$$
$$-\nabla u(x,t) \cdot \vec{n} = j(x,t) \qquad \text{on } \Gamma_N = \partial\Omega \setminus \Gamma_D, \tag{3}$$
$$u(x,t_0) = u_0(x) \qquad \text{at } t = t_0 \tag{4}$$

with the following choices applied:

$$q(u) = \eta u^2 \tag{5}$$
$$f = 0 \tag{6}$$
$$\Gamma_D = \{x \in \partial\Omega \mid x_0 = 0\} \tag{7}$$
$$g(x,t) = \sin(2\pi t) \prod_{i=1}^{d-1} \sin(\pi x_i)^2 \sin(10\pi x_i)^2 \tag{8}$$
$$j(x,t) = 0 \tag{9}$$
$$u_0(x) = g(x,0) = 0 \tag{10}$$
$$t_0 = 0. \tag{11}$$

The code to this exercise can be recompiled individually **in your build directory** by typing make:

```
[user@localhost]$ cd release-build/dune-pdelab-tutorials/
   ↪ tutorial03/exercise/task
[user@localhost]$ make
```

The structure of the code is very similar to the previous tutorials, it consists of the following files:

- `exercise03.cc` – main program,
- `driver.hh` – driver to solve problem on a gridview, hard-codes $t_0 = 0$
- `problem.hh` – problem parameter class, definitions of $q(u)$, f, Γ_D, Γ_N, g and j
- `nonlinearheatfem.hh` instationary spatial and temporal local operators $r(u,v,t)$ and $m(u,v,t)$ respectively .

As in the previous exercises you can control most of the settings through the ini-file `tutorial03.ini`. Get an overview of the configurable settings, compile and run `exercise03`.

The program writes output with the extension `pvd`. This is one of several ways to write VTK output for the instationary case, c.f. the documentation of the `tutorial03`. The `pvd`-file can be visualized by ParaView and it consists of a collection of the corresponding `vtu`-files. One big advantage of this approach is that the physical time can be printed out. This can be achieved by using the "Annotate-TimeFiler" in ParaView.

Exercise 2 Making Discretizations Easily Exchangeable

Step 1: Switching to the linear heat equation For the rest of the exercise we want to consider the linear heat equation. Therefore the reaction term has to be set to $q = 0$. Recompile and rerun `exercise03.cc` and investigate the difference to the nonlinear reaction term.
Hint for the rest of the exercise: For different runs of the simulation you can change the output filename in `tutorial03.ini`.

Since the initial problem (5)–(11) was nonlinear, Newton's method is used to solve the discretized equations. For the linear case it is sufficient to use the class `StationaryLinearProblemSolver`. Search in the driver for the lines starting with

```
   typedef Dune::PDELab::Newton<IGO,LS> PDESOLVER;
   PDESOLVER pdesolver(igo,ls);
   ...
```

and change to the `StationaryLinearProblemSolver`.

It has the same template arguments as the Newton solver. Give the instance of the class `StationaryLinearProblemSolver` also the name `pdesolver`. If you have problems with the construction of this solver consider for example the code in the driver of `tutorial00`. Compile and run again. The program reports the status of the solver. Get used to these different two outputs.

As a next step we want to use two spatial discretizations, i.e. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ elements. The degree of the spatial discretization can be changed in the ini-file. Currently $\mathcal{Q}_1$ elements are used. Please change to $\mathcal{Q}_2$ elements and rerun the simulation.

Step 2: Arbitrary one-step schemes We want to examine the numerical solution under three different time discretization schemes – Implicit Euler, Crank-Nicolson and Fractional-Step-θ. In order to change the time discretization scheme you will have to go to the file `driver.hh` and search for the line

```
   Dune::PDELab::Alexander2Parameter<RF> pmethod;
```

Change this to use the `Dune::PDELab::ImplicitEulerParameter<RF>`, compile and rerun the simulation. The program reports the progress of the time stepping and the method used. Convince yourself that you are using indeed the Implicit Euler. The other two time stepping methods can be applied similarly.

Note that there is no special one step parameter class for Crank-Nicolson. Crank-Nicolson is however the special case of the one step θ scheme with $\theta = 0.5$. You can create a parameter object for Crank-Nicolson with:

```
Dune::PDELab::OneStepThetaParameter<RF> pmethod(0.5);
```

A parameter object for the Fractional Step θ scheme can be created with:

```
Dune::PDELab::FractionalStepParameter<RF> pmethod;
```

Step 3: Different Initial and Boundary Conditions

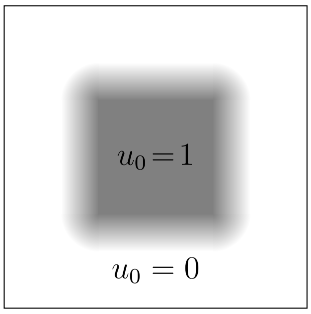

Figure 1: Initial conditions

Consider the initial and boundary conditions (5)–(11) modified as follows:

$$\Gamma_D = \emptyset \tag{12}$$

$$u_0(x) = g(x,0) = \prod_{i=0}^{d-1} \min\{1, \max\{0, \tilde{f}(x_i)\}\} \tag{13}$$

$$\tilde{f}(\xi) := 0.5 - 8(|\xi - 0.5| - 0.25)$$

$$j(x,t) = 0. \tag{14}$$

The initial condition given by u_0 models a block of constant 1 concentration in the middle, constant 0 concentration at the border and some linear decrease in between.

On a 16×16 or finer grid the initial values can be represented exactly by $\mathcal{Q}_1$ and $\mathcal{Q}_2$ finite elements. The exact solution will instantly become smooth and tend toward the mean over time. A computed solution is only an approximation, and may show different behavior. Most often it may take a long time for the solution to become smooth and, depending on the time stepping scheme used, there are spikes oscillating from one time step to the next.

Please implement the initial and boundary conditions (12)–(14). When implementing u_0 you may be tempted to use the function `abs()`. This is wrong, `abs()` is a function from the C-library to compute absolute values for *integers*. If a floating point value is plugged in, it is truncated to an integer. The correct way is to use `std::abs()` instead.

Compile and run your program. Remember that the grid needs 16×16 elements for the $\mathcal{Q}_1/\mathcal{Q}_2$ elements to resolve the initial condition exactly. What happens to the interpolated initial condition if you use a coarser mesh?

Step 4: Investigate Maximum Principle With these preparations done, it is now time to actually check how the different discretizations perform. Run your program to produce some output that you can examine in ParaView. Change the settings in the ini-file to a 64×64 grid and the time step size `<dt>` $= 1/64 = 0.015625$. Run the simulation until `<tend>` $= 4 \cdot \mathrm{dt}$. When examining the solution in ParaView, apply the "Warp by Scalar" filter to get an image distorted into the third dimension according to the values of the solution.

After one time step, the solution computed by $\mathcal{Q}_1$ finite elements with Implicit Euler time stepping should be completely smooth. The same goes for $\mathcal{Q}_2$ with Implicit Euler.

With both Crank-Nicolson and Fractional Step θ, both with $\mathcal{Q}_1$ and $\mathcal{Q}_2$ the solution should be quite non-smooth, i.e. there should be some edges visible. The Fractional Step θ scheme should be smoother than Crank-Nicolson.

Try to run the simulation with smaller time steps. How small do you need to make the time steps to get smooth solutions with Fractional Step θ?

Can you get the Crank-Nicolson scheme to produce smooth solutions as well?

Exercise 3 Time Dependent g and j

Consider now the initial and boundary conditions (5)–(11) time dependent:

$$\Gamma_D = \{x \in \partial\Omega \mid x_0 = 0\} \tag{15}$$

$$g(x,t) = t/10 \tag{16}$$

$$u_0(x) = g(x,0) \tag{17}$$

$$j(x,t) = -(0.5 + \cos(t)/2). \tag{18}$$

Incorporating the time dependence into the functions g and j is easy even if they don't have the time variable as an argument. The problem parameter class possesses the member variable `t` and the member function

```
    void setTime (Number t_)
    {
      t = t_;
    }
```

by means of which the correct time is always available. Please implement the conditions (15)–(18), compile and rerun the program. You might also want to increase the final time of the simulation. Examine the results in ParaView with the "Warp by Scalar" filter.

A short note on time dependent Γ_D and Γ_N: In principle it is possible to implement time dependent Γ_D and Γ_N the same way as for g and j. But for conforming spatial discretizations there is an important limiting assumption, namely that the type of boundary conditions do not change over a time step.