Exercises for Tutorial04

Finite Elements for the Wave Equation

In this exercise you will:

- Work on the problem presented in `tutorial04`.
- Try various time integrators, in particular the Crank-Nicolson method.
- Explore polynomial degrees greater than 2 (remember to change the blocking to `none`!).
- Implement your own local operator that switches to an elliptic projection method (see the tutorial notes and Eriksson et al. [1])
- Check the energy conservation property in the numerical scheme.

Exercise 1 Getting to Know the Code

The code of **exercise04** solves the wave equation formulated as a first order in time system. As already explained in `tutorial04`, we can write the wave equation as a system of two equations by substituting $u_0 = u$ and introducing $u_1 = \partial_t u_0 = \partial_t u$:

$$\partial_t u_1 - c^2 \Delta u_0 = 0 \qquad \text{in } \Omega \times \Sigma, \tag{1a}$$
$$\partial_t u_0 - u_1 = 0 \qquad \text{in } \Omega \times \Sigma, \tag{1b}$$
$$u_0 = 0 \qquad \text{on } \partial\Omega, \tag{1c}$$
$$u_1 = 0 \qquad \text{on } \partial\Omega, \tag{1d}$$
$$u_0 = q \qquad \text{at } t = 0, \tag{1e}$$
$$u_1 = w \qquad \text{at } t = 0. \tag{1f}$$

Since $u_0 = u = 0$ on the boundary we also have $\partial_t u = u_1 = 0$ on the boundary. Alternatively, one may also simply omit the boundary condition on u_1.

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

```
[user@localhost]$ cd release-build/dune-pdelab-tutorials/
  ↪ tutorial04/exercise/task
[user@localhost]$ make clean && make
```

Note that in contrast to earlier exercises, three different executables are built from the same source files, one each for 1D, 2D and 3D, in order to reduce compile times and compiler memory usage.

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

- `exercise04.ini` – holds parameters read by various parts of the code which control the execution,
- `exercise04.cc` – main program,
- `driver.hh` – instantiates the necessary PDELab classes for solving a linear instationary problem and finally solves the problem,
- `wavefem.hh` – contains the local operator classes `WaveFEM` and `WaveL2` realizing the spatial and temporal residual forms $r(u, v, t)$ and $m(u, v, t)$, respectively.

As in the previous exercises you can control most of the settings through the ini-file `exercise04.ini`. Get an overview of the configurable settings, compile and run `exercise04`.

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 tutorial03. The `pvd`-file can be visualized by ParaView and consists of a collection of the corresponding `vtu`-files. In order to visualize a 1D solution, one can apply the "Plot Over Line " filter. Note that our solution is always given by u_0.

Exercise 2 Try various time integrators, in particular the Crank-Nicolson method

We want to examine the numerical solution under different time discretization schemes, eg. Implicit Euler or Crank-Nicolson. 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;
```

Recall the previous `exercise03` and change the one step θ scheme in a way that corresponds to the Crank-Nicolson method.

Note that you can compare the solutions in ParaView. To do that you have to rename the second solution in `exercise04.ini` and use the "Append Attributes" filter before "Plot Over Line". Do not forget to change the parameter `torder` in `exercise04.ini` in order to get the correct scheme.

Remark You can decide which dimension to investigate. A 2D simulation will give you nicer pictures at the cost of longer run times.

Exercise 3 Explore polynomial degrees greater than 2 by changing the blocking to `none`.

Step 1: Find the place where your Local Finite Element Maps are created and make it possible to use polynomial degree 3

```
Dune::PDELab::PkLocalFiniteElementMap<GV,DF,double,deg> FEM;
```

Compile your program and check the results.

Note that `deg` is a static template parameter.

The program should throw an assertion error during compilation.

Step 2:

Before we start please have a look at the description of the driver in `tutorial04` ↪ and read the part that describes the specification of an ordering when creating product spaces. Using fixed block structure in ISTL requires that the number of degrees of freedom per entity is constant for each geometry type. This is true for polynomial degrees one and two but not for higher polynomial degree!

To avoid segfaults you need to change

```
using VBE = Dune::PDELab::istl::VectorBackend<Dune::PDELab::
   ↪ istl::Blocking::fixed>;
```

to

```
using VBE = Dune::PDELab::istl::VectorBackend<Dune::PDELab::
   ↪ istl::Blocking::none>;
```

in `driver.hh`.

Exercise 4 Changing the Local Operator

Now consider the elliptic projection as in Eriksson et al. [1], namely applying the Laplacian to equation (1b)

$$-\Delta\partial_t u_0 + \Delta u_1 = 0, \tag{2}$$

which has the advantage of energy conservation but requires additional smoothness properties.

The main work is now to change the local operators given in the file `wavefem.hh`. This is done in several steps:

- Copy the file `wavefem.hh` to a new file and rename it.
- Rename the local operators in the new file, e.g. change `WaveL2` to `WaveElip`.
- Include your new file in `exercise04.cc` and change the types of `LOP` and `TLOP` in `driver.hh`.

```
// Make instationary grid operator
double speedofsound = ptree.get("problem.speedofsound"
   ↪ ,1.0);
using LOP = WaveFEM<FEM>;
LOP lop(speedofsound);
using TLOP = WaveL2<FEM>;
TLOP tlop;
```

After these preparations you need to implement the following change:

$$\partial_t u_0 - u_1 = 0 \Rightarrow \Delta\partial_t u_0 - \Delta u_1 = 0.$$

- In the spatial local operator, change:

  $$-u_1 = 0 \Rightarrow -\Delta u_1 = 0$$

  See how it is done for the Δu_0, we recall the corresponding part of `WaveFEM::` ↪ `alpha_volume()`:

```
// integrate both equations
RF factor = ip.weight() * geo.integrationElement(ip.
   ↪ position());
for (size_t i=0; i<lfsu0.size(); i++) {
        r.accumulate(lfsu0,i,c*c*(gradu0*gradphi[i][0])*
           ↪ factor);
        r.accumulate(lfsu1,i,-u1*phihat[i]*factor);
}
```

- In the temporal local operator, change:

  $$\partial_t u_0 = 0 \Rightarrow \Delta\partial_t u_0 = 0$$

  See how it is done for the Δu_0, we recall the corresponding part of `WaveL2::` ↪ `alpha_volume()`:

```
// integrate u*phi_i
for (size_t i=0; i<lfsu0.size(); i++) {
        r.accumulate(lfsu0,i,u1*phihat[i]*factor);
        r.accumulate(lfsu1,i,u0*phihat[i]*factor);
}
```

- Do not forget to update the `jacobian_volume()` methods accordingly. As the problem is linear, this should not be too difficult.

Exercise 5 Energy conservation

If we multiply (1a) by u_1 and (2) by u_0 and add, the terms $-(\Delta u_0, u_1)$ and $(\Delta u_1, u_0)$ cancel out, leading to the conclusion that the energy

$$E(t) = \|u_1\|^2 + \|\nabla u_0\|^2$$

is constant in time.

Your task is to check the energy conservation for the elliptic projection and Crank-Nicolson in time. You can use the following PDELab utilities:

```
Dune::PDELab::SqrGridFunctionAdapter
Dune::PDELab::integrateGridFunction
Dune::PDELab::DiscreteGridFunctionGradient
Dune::PDELab::SqrGridFunctionAdapter
```

If you have problems with this task check the online documentation `https://www.dune-project.org/doxygen/pdelab/master/` or the solution.

Additional task

If you are done with these exercises, you can play with the initial conditions. Use the following setting in one space dimension:

- Change the initial conditions to $\sin(2x)$ (implement it as a lambda function)
- Change the domain size to $[0, \pi]$

References

[1] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational Differential Equations.* Cambridge University Press, 1996. `http://www.csc.kth.se/~jjan/private/cde.pdf`.