# DOpE - Deal Optimization Environment

Michael Geiger[*], Christian Goll[†], Thomas Wick[‡], Winnifried Wollner[§]

February 29, 2012

[*]University of Heidelberg, michael.geiger@iwr.uni-heidelberg.de
[†]University of Heidelberg, christian.goll@iwr.uni-heidelberg.de
[‡]University of Heidelberg, thomas.wick@iwr.uni-heidelberg.de
[§]University of Hamburg, winnifried.wollner@math.uni-hamburg.de

# Contents

# 1 Introduction

In this report, we describe the DOpE *Deal Optimization Environment* project. Originally, the project was initiated in the year 2009 at the University of Heidelberg (Germany) in the numerical analysis group of Rolf Rannacher.

The *Deal Optimization Environment* (DOpE) project is mainly based on the *deal.II* finite element library which has been developed initially by W. Bangerth, R. Hartmann, and G. Kanschat. In addition, some algorithmic aspects to compute optimization problems in which the state is given in terms of a partial differential equations are similar to algorithms of the Gascoigne/RoDoBo project, which was initiated by Roland Becker, Dominik Meidner, and Boris Vexler. Both projects have also been initiated at the University of Heidelberg.

The aim of DOpE is to provide a software toolkit to solve forward PDE problems as well as optimal control problems constrained by PDE. The solution of a broad variety of PDE is possible in deal.II as well, but DOpE concentrates on a unified approach for both linear and nonlinear problems by interpreting every PDE problem as nonlinear and applying a Newton method to solve it. While deal.II leaves much of the work and many decisions to the user, DOpE intends to be user-optimized by delivering prefabricated tools which require from the user only adjustments connected to his specific problem. The solution of optimal control problems with PDE constraints is an innovation in the DOpE framework. The focus is on the numerical solution of both stationary and nonstationary problems which come from different application fields, like elasticity and plasticity, fluid dynamics, and fluid-structure interactions.

At the present stage, we are able to provide the following features and problems:

- Computation of forward problems in 1d, 2d, and 3d.
- Solution of stationary and nonstationary forward problems
- Various timestepping schemes (based on finite differences), such as forward Euler, backward Euler, Crank-Nicolson, shifted Crank-Nicolson, and Fractional-Step-$\Theta$ scheme.
- Various finite elements taken from deal.ii.
- Computation of stationary optimal control problems.
- Solving of standard Laplace equation, Navier-Stokes, and fluid-structure interaction.

In the present manual we give a short introduction to all example programs which can be found currently in DOpE. The organization of the examples is as follows: we

distinguish between the mere solution of a PDE and the solution of an optimal control problem, and in each case we differentiate further between stationary and nonstationary problems. This leads to the following structure (we list problems treated in the examples in brackets):

- solution of PDE problem
  - stationary PDE (Laplace equation, Stokes equations, FSI problems, elasticity and plasticity problems)
  - nonstationary PDE (Navier-Stokes equations, FSI problem, Black-Scholes equation)
- solution of PDE constrained optimal control problem
  - stationary constraining PDE
  - nonstationary constraining PDE

The seventh stationary PDE example 3.1.7 (solution of the Laplace equation) is implemented analogously to the corresponding tutorial step in deal.II. By comparing the respective program codes, deal.II users should be able to catch the differences between the two software packages. In later examples we also provide algorithms for 'goal-oriented' automatic mesh adaption and error analysis.

**Comment on deal.II**

The deal.II library is open source and can be downloaded from `http://www.dealii.org/`. On this homepage, one also finds lots of further information on deal.II as well as an extensive tutorial where many features of deal.II are discussed in a well-documented example framework. In order to use DOpE, it is highly recommendable to be roughly acquainted with deal.II.

# 2 Example Handling, Creating new Examples

To implement new examples or to use existing examples from the library for own research, the user can simply copy an existing example. In this new example, own code and changes can be compiled.

Adding a new regression test to the repository needs some concentration of the user. In the following, we explain how to do this and what *must* be taken into account:

```
cp -r Example1 ExampleNew
cd ExampleNew
rm -r CVS
cd Test
rm -r CVS
```

It is important to remove the repository information, which is stored in the directory CVS.

Then, we have to consider the following step. In order to run the regression tests, we start the executable file of the present example. This executable comes from the Makefile. For example in the Makefile of Example 3.1.1,

```
target   = $(BINDIR)/DOpE-PDE-StatPDE-Example1-$
           (dope_dimension)d-$(deal_II_dimension)d
```

and concretely:

```
DOpE-PDE-StatPDE-Example1-2d-2d
```

The aforementioned expression is the name of the executable that is required to run the regression test. Hence, in

```
PDE/StatPDE/Example1/Test> emacs test.sh
```

you find two times the line

```
echo "Running Program
     ../../../../../bin/DOpE-PDE-StatPDE-Example1-2d-2d test.prm"
     (../../../../../bin/DOpE-PDE-StatPDE-Example1-2d-2d test.prm 2>&1)
     > /dev/null
```

It is really important when copying an existing example, to change this information. First, we have to change the target in the Makefile

```
target   = $(BINDIR)/DOpE-PDE-StatPDE-ExampleNew-$
           (dope_dimension)d-$(deal_II_dimension)d
```

Second, we have to replace Example1 by ExampleNew such that

```
echo "Running Program
     ../../../../../bin/DOpE-PDE-StatPDE-ExampleNew-2d-2d test.prm"
     (../../../../../bin/DOpE-PDE-StatPDE-ExampleNew-2d-2d test.prm 2>&1)
     > /dev/null
```

Please, also change the dimension parameters

```
-2d-2d
```

Now, the user is prepared to change any information in the

```
main.cc, localpde.h, functionals.h, localfunctional.h, etc
```

# 3 Examples for PDE Solution

## 3.1 Stationary PDEs

### 3.1.1 Stationary Stokes Equations

#### General problem description

In this example we consider the stationary incompressible Stokes equation . Here, we use the symmetric stress tensor which has a little consequence when using the do-nothing outflow condition. In strong formulation we have

$$-\frac{1}{2}\nabla \cdot (\nabla v + \nabla v^T) + \nabla p = f \tag{3.1}$$
$$\nabla \cdot v = 0$$

on the domain $\Omega = [-6, 6] \times [0, 2]$. We choose for simplicity $f = 0$.

#### Program structure

In all examples, the whole program is split up into several files for the sake of readability. These files are always denoted in the same way, so we only have to explain the general structure in this first example, whereas in the following examples, we will only point out differences to the current one. The content of the single files will be described in more detail below.

If we do not use one of the standard grids given in the deal.II library, we can read a grid from an input file. In our example, the domain $\Omega = [-6, 6] \times [0, 2]$ is given in the *channel.inp* file, where all nodes, cells and boundary lines are listed explicitly and the boundary is divided into disjoint parts by attributing different colors to the boundary lines.

Certain parameters occurring during the solution process, e.g. error tolerances or the maximum number of iterations in an iterative solution procedure, are fixed in a parameter file called *dope.prm*. This parameter file comprises several subsections corresponding to different solver components.

In the *functionals.h* file we declare classes for different scalar quantities of interest (described mathematically as functionals) which we want to evaluate during the solution

process.

The *localfunctional.h* file is relevant only if we want to solve an optimal control problem. In this case, it contains the cost functional, whereas the file is not needed for the forward solution of PDEs. We will get back to this later in the context of optimal control problems.

All information about the PDE problem (in the optimal control case about the constraining PDE) is included in the *localpde.h* file. In a class called `LocalPDE`, we build up the cell equation, the cell matrix and cell righthand side as well as the boundary equation, boundary matrix and boundary righthand side. Later on, the integrator collects this local information and creates the global vectors and matrices.

The most important part of each example is the *main.cc* file which contains the `int main()` function. Here we create objects of all classes described above and actually solve the respective problem.

**The** functionals.h **file**

Here, we declare all quantities of interest (functionals), e.g. point values, drag, lift, mean values of certain quantities over a subdomain etc.
Each of these functionals is declared as a class of its own, but in DOpE all classes are derived from a so-called `FunctionalInterface` class.
In the current example we declare functionals for point values of the velocity and for the flux at the outflow boundary of the channel.

**The** localpde.h **file**

The *LocalPDE* is derived from a `PDEInterface` class. It comprises several functions which build up the cell and boundary equations, matrices and righthand sides. The weak formulation of problem (3.1) with $f = 0$ is

$$\frac{1}{2}(\nabla v, \nabla \varphi) + \frac{1}{2}(\nabla v^T, \nabla \varphi) - (p, \nabla \cdot \varphi) + (\nabla \cdot v, \psi) = 0. \tag{3.2}$$

This problem is vector valued, i.e. the velocity variable $v$ has two components and the pressure variable $p$ is a scalar. For the implementation, we use a vector valued solution variable with three components, where the distinction between velocity and pressure is done by use of the deal.II `FEValuesExtractors` class.
Furthermore, in DOpE we always interpret the problems in the context of a Newton method. Usually, a PDE in its weak formulation is given as

$$a(u; \varphi) = f(\varphi).$$

The lefthand side is implemented in the `CellEquation` function, the righthand side is implemented in the `CellRightHandSide` function (which is unused in this example, because $f = 0$).
To apply Newton's method, this problem is linearized: on the lefthand side, we have the derivative of the (semilinear) form $a(\cdot;\cdot)$ with respect to the solution variable $u$, and the righthand side is the residual of the weak formulation:

$$a'_u(u; u^+, \varphi) = -a(u; \varphi) + f(\varphi).$$

In the `CellMatrix` function, we implement the following matrix $A$ as representation of the derivative on the lefthand side:

$$A = (a'_u(u; \varphi_i, \varphi_j))_{j,i=1}^N$$

with the number $N$ of the degrees of freedom. Similarly, the `CellEquation` contains the vector

$$a = a(u; \varphi_i)_{i=1}^N,$$

and the `CellRightHandSide` in the case $f \neq 0$ would contain a vector

$$\tilde{f} = (f; \varphi_i)_{i=1}^N.$$

The system of equations which is then actually solved is

$$A\tilde{u}^+ = -a + \tilde{f}.$$

Because of the linearity of equation (3.2), there is almost no difference between the two functions.

*At this point, it is important to note that DOpE interprets any given problem as a nonlinear one which is solved by Newton's method; the special case of linear problems is included into this general framework.*

**The** main.cc **file**

First of all, several header files have to be included that are needed during the solution process. We divide these includes into blocks corresponding to DOpE headers, deal.II headers, C++ headers and header files of the example itself (like the ones mentioned above).
Furthermore, we define names for certain objects via `typedef` which act as abbreviations in order to keep the code readable. In our case, these are `OP`, `INTEGRATOR`, `LINEARSOLVER`, `NLS` and `SSOLVER`.
In the `int main()` function, we first create a possibility to read the parameter values from the *dope.prm* file. Then there are several standard steps for finite element codes like

- definition of a triangulation and create a grid object (which we read from the *channel.inp* file)
- creation of finite element objects for the state and the control and of quadrature formula objects

and in addition, we

- create objects of the `LocalPDE` class and of the different functional classes declared in the *functionals.h* file (including an object of the `LocalFunctional` class, even if we do not regard an optimal control problem here).

*REMARK:*
Up to now we have to create a pseudo time even for stationary problems. The `MethodOfLines_StateSpaceTimeHandler` object (`DOFH`) which is needed for the initialization of `OP` requires a vector in which timepoints are specified. However, this is again merely a dummy variable, for we do not actually apply a time stepping method in the stationary case. This will also be removed in future versions of DOpE.

Before we initialize the `SSolver` object and actually solve the problem, we have to set the correct boundary conditions. Via the `compmask` vector, we ensure that the boundary conditions are set only for the velocity components of our solution vector. We set homogeneous Dirichlet values at the upper and lower boundaries of the channel. The inflow is described by a parabolic profile at the left boundary (the corresponding function class is declared in the *myfunctions.cc* file), whereas we do not prescribe anything at the outflow boundary (so-called do-nothing condition).
The output of the program (the two functional values) is rather unspectacular; as the problem is linear, the solution is computed within one Newton step.

### 3.1.2 Stationary FSI with INH Material

**General problem description**

In this example we consider a simple stationary FSI problem. The fluid is given as an incompressible Newtonian fluid modelled by the Stokes equation. Here, we use the symmetric stress tensor which has a little consequence when using the do-nothing outflow condition. The computation domain is $\Omega = [-6, 6] \times [0, 2]$ and we choose for simplicity $f = 0$.

The fluid reads:

**Problem 3.1.1 (Variational fluid problem, Eulerian framework)** *Find* $\{v_f, p_f\} \in \{v_f^D + V\} \times L_f$*, such that,*

$$\begin{aligned}(\rho_f v \cdot \nabla v_f, \phi^v)_{\Omega_f} + (\sigma_f, \nabla \phi^v)_{\Omega_f} &= \langle n_f \cdot g_s^\sigma, \phi^v \rangle_{\Gamma_i} \quad && \forall \phi^v \in V_f, \\ (\text{div } v_f, \phi^p)_{\Omega_f} &= 0 && \forall \phi^p \in L_f.\end{aligned}$$

*The Cauchy stress tensor* $\sigma_f$ *is given by*

$$\sigma_f := -p_f I + \rho_f \nu_f (\nabla v_f + \nabla v_f^T), \tag{3.3}$$

*with the fluid's density* $\rho_f$ *and the kinematic viscosity* $\nu_f$*. By* $n_f$ *we denote the outer normal vector on* $\Gamma_i$ *and by* $g_f^\sigma$ *is a function which describes forces acting on the interface. These will be specified in the context of fluid-structure interaction models.*

We define:

$$\hat{T} := \text{id} + \hat{u}, \quad \hat{F} := I + \hat{\nabla}\hat{u}, \quad \hat{J} := \det(I + \hat{\nabla}\hat{u}).$$

The structure equations are given by incompressible neo-Hookean material

**Problem 3.1.2 (Incompressible neo-Hookean Model (Lagrangian))**

$$\begin{aligned}(\hat{J}_s \hat{\sigma}_s \hat{F}_s^{-T}, \hat{\nabla}\hat{\phi}^v)_{\hat{\Omega}_s} &= \langle \hat{J}_s \hat{n}_s \cdot \hat{g}_s^\sigma \hat{F}_s^{-T}, \hat{\phi}^v \rangle_{\hat{\Gamma}_i} \quad && \forall \hat{\phi}^v \in \hat{V}_s \\ (\hat{v}_s, \hat{\phi}^u)_{\hat{\Omega}_s} &= 0 && \forall \hat{\phi}^u \in \hat{V}_s, \\ (det\, \hat{F}_s - 1, \hat{\phi}^p)_{\hat{\Omega}_s} &= 0 && \forall \hat{\phi}^p \in \hat{L}_s,\end{aligned}$$

*where* $\rho_s$ *is the solid's density,* $\mu_s$ *the Lamé coefficient,* $\hat{n}_s$ *the outer normal vector at* $\hat{\Gamma}_i$*,* $\hat{g}_s^\sigma$ *the force on the interface and with*

$$\hat{\sigma}_s := -\hat{p}_s I + \mu_s (\hat{F}_s \hat{F}_s^T - I).$$

The resulting FSI problem is then given by:

**Problem 3.1.3 (Stationary Fluid-Structure Interaction (ALE))**

$$\begin{aligned}(\hat{J}\rho_f \hat{F}^{-1}\hat{v} \cdot \hat{\nabla}\hat{v}, \hat{\phi}^v)_{\hat{\Omega}_f} + (\hat{J}\hat{\sigma}_f \hat{F}^{-T}, \hat{\nabla}\hat{\phi}^v)_{\hat{\Omega}_f} & \\ + (\hat{J}\hat{\sigma}_s \hat{F}^{-T}, \hat{\nabla}\hat{\phi}^v)_{\hat{\Omega}_s} = 0 \quad & \forall \hat{\phi}^v \in \hat{V}, \\ (\hat{v}, \hat{\phi}^u)_{\hat{\Omega}_s} + (\alpha_u \hat{\nabla}\hat{u}, \hat{\nabla}\hat{\phi}^u)_{\hat{\Omega}_f} = 0 \quad & \forall \hat{\phi}^u \in \hat{V}, \\ (\widehat{div}(\hat{J}\hat{F}^{-1}\hat{v}_f), \hat{\phi}^p)_{\hat{\Omega}_f} + (\hat{J} - 1, \hat{\phi}^p)_{\hat{\Omega}_s} = 0 \quad & \forall \hat{\phi}^p \in \hat{L},\end{aligned}$$

**Program description**

There is not much difference to the prior example, so we will keep our remarks rather short. The *functionals.h, localfunctional.h* and *myfunctions.cc* files did not change at all, so we simply refer to the corresponding sections in the last example.
In the *localpde.h* file, all functions of the `LocalPDE` class have to be adjusted to the current FSI problem. This only makes the equations and matrices a little more complicated, and our solution vector now consists of five components (two velocity components of the fluid, the pressure component, and two additional displacement components for the structure variables). Otherwise, everything is analogous to the former example.
In the *main.cc* we only have to add two components to the `compmask` vector and prescribe boundary conditions for the structure variables. Apart from that, we define objects for the same classes as before that are even named equally and use the same solvers.
Again, the solution is reached within one Newton step, and all we see from the program output is the values of the functionals.

### 3.1.3 Stationary FSI with STVK Material

**General problem description**

This example is an extension the previous one. We solve an stationary FSI problem either with INH material (see Problem definition before) or St. Venant Kirchhoff material STVK:

**Problem 3.1.4 (Compressible Saint Venant-Kirchhoff, Lagrangian framework)** *Find* $\{\hat{v}_f, \hat{u}_f\} \in \{\hat{v}_f^D + \hat{V}\} \times \{\hat{u}_f^D + \hat{V}\}$, *such that*

$$\begin{aligned} (\hat{J}_s \hat{\sigma}_s \hat{F}_s^{-T}, \hat{\nabla}\hat{\phi}^v)_{\hat{\Omega}_s} &= \langle \hat{J}_s \hat{n}_s \cdot \hat{g}_s^\sigma \hat{F}_s^{-T}, \hat{\phi}^v \rangle_{\hat{\Gamma}_i} \qquad && \forall \hat{\phi}^v \in \hat{V}_s \\ (\hat{v}_s, \hat{\phi}^u)_{\hat{\Omega}_s} &= 0 && \forall \hat{\phi}^u \in \hat{V}_s, \end{aligned} \tag{3.4}$$

*where* $\rho_s$ *is the density of the structure,* $\mu_s$ *and* $\lambda_s$ *the Lamé coefficients,* $\hat{n}_s$ *the outer normal vector at* $\hat{\Gamma}_i$, $\hat{g}_s^\sigma$ *some forces on the interface. The properties of the STVK material is specified by the constitutive law*

$$\hat{\sigma}_s := \hat{J}^{-1} \hat{F} (\lambda_s (tr\hat{E}) I + 2\mu_s \hat{E}) \hat{F}^{-T}. \tag{3.5}$$

Often, the elasticity properties of structure materials is characterized by Poisson's ratio $\nu_s$ ($\nu_s < \frac{1}{2}$ for compressible materials) and the Young modulus $E$ . The relationship to the Lamé coefficients $\mu_s$ and $\lambda_s$ is given by:

$$\nu_s = \frac{\lambda_s}{2(\lambda_s + \mu_s)}, \qquad E = \frac{\mu_s(\lambda_s + 2\mu_s)}{(\lambda_s + \mu_s)}. \tag{3.6}$$

The whole equation system is solved on the benchmark configuration domain. For details on parameters and geometry, we refer to the numerical FSI benchmark proposal from Hron and Turek [2006].

The code is established by computing the stationary FSI benchmark example FSI 1 with the following values of interest: $x$-displacement, $y$-displacement, drag, and lift.

**Program description**

Compared to the two former examples, there are some differences which we will briefly discuss in the following. First of all, the problem is nonlinear in contrast to the former ones. We work on a different domain (given in the *benchfst0100tw.inp* file), namely a channel with a cylinder put at half height near the inflow boundary; further *.inp* files yield the possibility to vary the domain.
Furthermore, in the *dope.prm* parameter file there are two additional subsections which are added only for the current problem. From the denotation of these subsections one can immediately see where in the code the parameters are used.
As we want to compute certain benchmark quantities, we have to regard corresponding functionals in the *functionals.h* file. The pressure at a point as well as the displacement in $x$- and $y$-directions are point values; furthermore we implement the drag and lift functionals (for which we need the additionally defined problem parameters).

As before, we build up the cell and boundary equations and matrices in the *localpde.h* file. Apart from using the additionally defined problem parameters and modelling compressible STVK material instead of INH material (which leads to changes in the weak formulation of the equations), there are no major differences to the corresponding file in the last example.
In the *main.cc* file, we have to include additional header files from the deal.II library concerning error estimation and grid refinement. Further on, everything is pretty much the same as in the last example, but we have to use the `SetBoundaryFunctionalColors` function of the `PDEProblemContainer` class to be able to compute drag and lift in the respective functional classes in *functionals.h*.
The main innovation in contrast to the preceding examples is the refinement of the grid combined with a simple error estimator given in the deal.II `KellyErrorEstimator` class. If we look at the output of our program, everything is computed several times (once on each refinement level). Furthermore, we see that several Newton steps are needed on each refinement level; this is due to the nonlinearity of the current problem.

### 3.1.4 Stationary Elasticity Benchmark

#### General problem description

In this example we consider the following benchmark problem from elasticity theory:

$$(\sigma(u), \varepsilon(\varphi)) = (g, \varphi)_{\Gamma_N}. \tag{3.7}$$

Here $\tilde{\Omega}$ is a quadratic domain with side length 200 mm, where a circular hole with radius 10 mm around the center is cut out. Using symmetries of the domain, we restrict our actual computational domain $\Omega$ to the upper left quarter of $\tilde{\Omega}$.
In the above equation, $\varepsilon(v) := \frac{1}{2}(\nabla v + \nabla v^T)$ is the symmetric strain tensor, and

$$\sigma(v) := 2\mu\varepsilon(v)^D + \kappa \cdot tr(\varepsilon(v))I$$

denotes the symmetric stress tensor. Here $\tau^D$ is the deviatoric part of a tensor $\tau$, in two dimensions defined as

$$\tau^D := \tau - \frac{1}{2}tr(\tau)I,$$

and the parameters $\mu$ and $\kappa$ are chosen as $\mu = 80193.800283$ resp. $\kappa = 190937.589172$.
The corner points of our computational domain are in anticlockwise order: $(0,0), (90,0)$, $(100,10), (100,100)$ and $(0,100)$. We prescribe homogeneous Dirichlet boundary conditions in $y$-direction between $(0,0)$ and $(90,0)$ (lower boundary part), homogeneous Dirichlet boundary conditions in $x$-direction between $(100,10)$ and $(100,100)$ (right boundary part), and we interpret the righthand side of equation (1) with $g = 450$ as a boundary condition between $(0,100)$ and $(100,100)$ (upper boundary part).
The goal of our computations is to match the following functional reference values taken from *E. Stein (editor), Error-controlled Adaptive Finite Elements in Solid Mechanics, Wiley (2003), pp. 386 - 387*:

| Functional | $u_1$ at $(90,0)$ | $\sigma_{22}$ at $(90,0)$ | $u_2$ at $(100,100)$ |
|---|---|---|---|
| Reference value | 0.021290 | 1388.732343 | 0.20951 |

| Functional | $u_1$ at $(0,100)$ | $\int_{(100,100)}^{(0,100)} u_2$ |
|---|---|---|
| Reference value | 0.076758 | 20.40344 |

#### Program description

From the previous examples we know how to read a grid from an *.inp* file. The grid of our current example comes from the above mentioned benchmark problem. The parameter file (*dope.prm*) contains the same variables as in the first two examples.
Apart from different point values of derivatives of the solution, we want to evaluate an integral over part of the boundary. This is newly implemented in *functionals.h*.
The (linear) elasticity equation is much simpler than the FSI problems we treated before, which can be seen in *localpde.h* where the `CellEquation` and `CellMatrix` are

implemented. In principle, everything is clear from the preceding examples.
The *main.cc* file does not contain any innovation, either. Here, we refine the grid globally instead of using an error estimator for local refinement.
The output of the program reflects again the linearity of the problem (only one Newton step is needed for solution).

### 3.1.5 Stationary Plasticity Benchmark

#### General problem description

Similar to the previous example, we consider the following benchmark problem from plasticity theory:

$$(\Pi(\sigma(u)), \varepsilon(\varphi)) = (g, \varphi)_{\Gamma_N}. \tag{3.8}$$

Here $\tilde{\Omega}$ is again the quadratic domain with a circular hole around the center cut out. Again, we restrict our actual computational domain $\Omega$ to the upper left quarter of $\tilde{\Omega}$ for reasons of symmetry.
We use the symmetric strain tensor $\varepsilon(v) := \frac{1}{2}(\nabla v + \nabla v^T)$, and the symmetric stress tensor $\sigma$ is defined as

$$\sigma(v) := 2\mu\varepsilon(v)^D + \kappa \cdot tr(\varepsilon(v))I$$

where $\tau^D$ is the deviatoric part of a tensor $\tau$, in two dimensions defined as

$$\tau^D := \tau - \frac{1}{2}tr(\tau)I.$$

The main difference with respect to the elastic case is the projection operator $\Pi$ in equation (1). It is defined as follows:

$$\Pi(\tau) = \begin{cases} \tau & |\tau^D| \leq \sigma_0 \\ \sigma_0|\tau^D|^{-1}\tau^D + \frac{1}{2}tr(\tau)I & |\tau^D| > \sigma_0 \end{cases}$$

In our computations, we choose $\sigma_0 = \sqrt{\frac{2}{3}} \cdot 450$, and the above parameters $\mu$ and $\kappa$ as $\mu = 80193.800283$ resp. $\kappa = 190937.589172$. The corner points of our computational domain are the same as before, and the boundary conditions are not altered, either.
The goal of our computations is to detect a subdomain in $\Omega$ where plastic behaviour occurs (compare *E. Stein (editor), Error-controlled Adaptive Finite Elements in Solid Mechanics, Wiley (2003), pp. 386 - 389*). This subdomain depends on the righthand side $g$ in equation (1) which we write as $g = \lambda \cdot p$ with $p = 100$ and $\lambda \in [1.5; 4.5]$.

#### Program description

The code of the current example is nearly identical to the code of the previous one. The only difference worth mentioning is the change of the equations which leads to different implementations of the `CellEquation, CellMatrix` and `BoundaryEquations` functions in *localpde.h*.
Furthermore, the elasticity equations solved in the last example are linear, whereas the plasticity equations are nonlinear; this difference is evident also from the output (here, we need several Newton steps until convergence).
The functionals that appear in the output yield additional information and are not required in the above problem setting. The subdomain with plastic behaviour we want to detect can be visualized from the *.vtk* files written to the `Results/Mesh` subfolders.

### 3.1.6 Stationary Stokes Equations with periodic BC

todo. short: this example shows how to use the constraintsmaker class by enforcing periodic boundary conditions in 2d.

### 3.1.7 Laplace Equation in 2D

In this example we consider the stationary incompressible Stokes equation. Here, we use the symmetric stress tensor which has a little consequence when using the do-nothing outflow condition. In strong formulation we have

$$\begin{aligned} -\nabla \cdot (\nabla v + \nabla v^T) + \nabla p &= f \\ \text{div } v &= 0 \end{aligned}$$

on the domain $\Omega = [-6, 6] \times [0, 2]$. We choose for simplicity $f = 0$.

### 3.1.8 Laplace Equation in 3D

Laplace in 3D on locally refined grids with Kelly estimator with iterative solver (CG without preconditioning)

## 3.2 Nonstationary PDEs

Until now, the DOpE provides various time-stepping schemes that are based on finite differences. Specifically, the user can choose between the

- Forward Euler scheme (FE), which is an explicit timestepping scheme. Here, one as to take into account that $k \leq ch^2$ where $k$ denotes the timestep size and $h$ the local mesh cell diameter.
- Backward Euler scheme (BE), which is an implicit timestepping scheme. It is strongly A-stable but only from first order and very dissipative. The BE-scheme is well suited for stationary numerical examples.
- Crank-Nicolson scheme (CN), which is of second order, A-stable, has very little dissipation but suffers from case to case from instabilities caused by rough initial- and/or boundary data. These properties are due to weak stability (it is not *strongly* A-stable).
- Shifted (or stabilized) Crank-Nicolson scheme (CN shifted), which is also of second order, but provides global stability.
- Fractional-step-$\theta$ scheme (FS). It has second-order accuracy and is strongly A-stable, and therefore well-suited for computing solutions with rough data.

### 3.2.1 Nonstationary Stokes Equations

In this example we consider the nonstationary incompressible Stokes equation.

**General problem description**

As in the stationary PDE Example 3 3.1.3, we use the symmetric fluid stress tensor. In strong formulation we deal with

$$\begin{aligned} \partial_t v - \nabla \cdot (\nabla v + \nabla v^T) + \nabla p &= f \\ div\, v &= 0 \end{aligned}$$

on time interval $I = [0, T]$ and the domain $\Omega = [-6, 6] \times [0, 2]$. For simplicity, we set $f = 0$.

As introduced earlier, we formulate the time stepping scheme as *One-step-$\theta$ scheme*, which are based on finite difference schemes. The time interval is given by $I = [0, T]$. Let $v^n, p^n$ and the time step $k = t^{n+1} - t^n$ be given. Find $v = v^{n+1}, p^{n+1}$ such that:

$$\begin{aligned} v - k\theta(\nabla \cdot (\nabla v + \nabla v^T) + \nabla p) &= k\theta f^{n+1} + k(1-\theta) f^{n+1} \\ &\quad + v^n + k(1-\theta)(\nabla \cdot (\nabla v^n + \nabla (v^n)^T) + \nabla p^n) \\ div\, v &= 0 \end{aligned}$$

In the case of the BE-scheme, $\theta = 1$, and the equation is reduced to

$$v - k\theta(\nabla \cdot (\nabla v + \nabla v^T) + \nabla p) = k\theta f^{n+1} + v^n$$
$$div\, v = 0$$

Note, that one should prefer a complete implicit treatment of the pressure $p$. Instead of using $\theta p^{n+1} + (1-\theta)p^n$, the pressure appears only with $\theta p^{n+1}$.

After discretization in time, the space is treated, as ususally, with a Galerkin finite element scheme, here based on the Taylor-Hood element $Q_2^c/Q_1^c$.

The variational formulation reads:

**Problem 3.2.1 (Backward Euler (BE) timestepping problem)** *Find* $v := v^{n+1} \in V$ *and* $p := p^{n+1} \in L$*:*

$$(v, \phi^v) + k\theta(\nabla v + \nabla v^T, \nabla \phi^v) - k(p, \nabla \cdot \phi^v) = k\theta(f^{n+1}, \phi^v) + (v^n, \phi^v)$$
$$(div\, v, \phi^p) = 0$$

for all suitable test functions $\phi^v, \phi^p \in V \times L$. The parameter $\theta$ is chosen as $\theta = 1$.

Derivation of the other timestepping problems is analogous.

**Specific features for solving nonstationary problems**

In the following, we explain in more detail the different member functions that are required to implement nonstationary equations.

```
void CellEquation (..., double scale, double scale_ico)
```

The two arguments are used to distigunish between explicit components and fully implicit components. For standard equations (such as the heat equation and the wave equation), there is no special treatment required needed.

However, solving the Navier-Stokes equations are multi-physics problems (like fluid-structure interaction), parts of the equations are treated with a fully implicit time-stepping scheme.

Thus, the argument

```
double scale
```

is used to indicate that the present term can be used for implicit/explicit or mixed discretization (such as time discretization with the Crank-Nicolson.

The other argument

```
double scale_ico
```

is used to indicate that the present term only is treated in a fully implicit manner. For example, the pressure term (which is of course a Lagrange multiplier of the incompressibility term of the fluid). It is recommended to treat this term in a time discretization in a fully implicit manner.

```
void CellMatrix (..., double scale, double scale_ico)
```

The directional derivatives of the state equation are implemented in the present function. As before, the last two parameters

```
double scale, double scale_ico
```

are used to distigungish between fully implicit are other behavior.

```
void CellTimeEquation (...)
```

This function is used to implement the time derivative in weak formulation

$$(\partial_t v, \phi)_\Omega.$$

This term is time discretized via

$$k^{-1}(v^n - v^{n-1}, \phi)_\Omega.$$

Here, it suffices to implement the term

$$(v^n, \phi)_\Omega,$$

because the already known term $v^{n-1}$ is automatically treated by the specific time stepping scheme.

In contrast to this behavior, the user has the possibility to write all terms of $\partial_t v$ explicitly. In this case, we use the

```
void CellTimeEquationExplicit (...)
```

and we write

$$(v^n - v^{n-1}, \phi)_\Omega.$$

This behavior is useful for multi-phyics problems where other solutions variables have to be considered around $\partial_t v$. The user should have a look in the second Example 3.2.2 for nonstationary problems for an illustration of this function.

Consequently, the directional derivatives of the cell terms are implemented in the corresponding matrix functions, i.e.,

```
void CellTimeMatrix (...), void CellTimeMatrixExplicit
```

### 3.2.2 Nonstationary FSI Problem

**General problem description**

In the present example, we solve a nonstationary fluid-structure interaction problem. The underlying equations are stated in the following:

**Problem 3.2.2 (Variational fluid-structure interaction framework)** *Find* $\{\hat{v}, \hat{u}, \hat{p}\} \in \{\hat{v}^D + \hat{\mathcal{V}}^0\} \times \{\hat{u}^D + \hat{\mathcal{V}}^0\} \times \hat{\mathcal{L}}$*, such that* $\hat{v}(0) = \hat{v}^0$ *and* $\hat{u}(0) = \hat{u}^0$*, for almost all time steps* $t$*, and*

$$
\begin{aligned}
(\hat{J}\hat{\rho}_f \partial_t \hat{v}, \hat{\psi}^v)_{\hat{\Omega}_f} + (\hat{\rho}_f \hat{J}(\hat{F}^{-1}(\hat{v} - \partial_t \hat{u}) \cdot \hat{\nabla})\hat{v}), \hat{\psi}^v)_{\hat{\Omega}_f} \\
+(\hat{J}\hat{\sigma}_f \hat{F}^{-T}, \hat{\nabla}\hat{\psi}^v)_{\hat{\Omega}_f} + (\hat{\rho}_s \partial_t \hat{v}, \hat{\psi}^v)_{\Omega_s} + (\hat{J}\hat{\sigma}_s \hat{F}^{-T}, \hat{\nabla}\hat{\psi}^v)_{\hat{\Omega}_s} \\
-\langle \hat{g}, \hat{\psi}^v \rangle_{\hat{\Gamma}_N} - (\hat{\rho}_f \hat{J}\hat{f}_f, \hat{\psi}^v)_{\hat{\Omega}_f} - (\hat{\rho}_s \hat{f}_s, \hat{\psi}^v)_{\hat{\Omega}_s} = 0 \quad \forall \hat{\psi}^v \in \hat{V}^0, \\
(\partial_t \hat{u} - \hat{v}, \hat{\psi}^u)_{\hat{\Omega}_s} + (\hat{\sigma}_g, \hat{\nabla}\hat{\psi}^u)_{\hat{\Omega}_f} - \langle \hat{\sigma}_g \hat{n}_f, \hat{\psi}^u \rangle_{\hat{\Gamma}_i} = 0 \quad \forall \hat{\psi}^u \in \hat{V}^0, \\
(\widehat{div}(\hat{J}\hat{F}^{-1}\hat{v}_f), \hat{\psi}^p)_{\hat{\Omega}_f} + (\hat{p}_s, \hat{\psi}^p)_{\hat{\Omega}_s} = 0 \quad \forall \hat{\psi}^p \in \hat{L},
\end{aligned}
$$

*with* $\hat{\rho}_f$*,* $\hat{\rho}_s$*,* $\nu_f$*,* $\mu_s$*,* $\lambda_s$*,* $\hat{F}$*, and* $\hat{J}$*. The stress tensors for the fluid and structure are implemented in* $\hat{\sigma}_f$*,* $\hat{\sigma}_s$*, and* $\hat{\sigma}_g$

**Code validation for fluid problems with ALE**

With the ALE code implemented in Example 3.1.3 it is possible to treat fluid problems as well as FSI computations. In the case of fluid problems the deformation gradient and its determinant become:

$$F := I, \quad \det F = J = 1.$$

The code is validated by the well-known fluid- and FSI benchmark problems. Here, the results have been summarized in the table below.

| Configuration | Date | Refinement | Time dis. | $k$ | $\Delta p$ | $F_D$ | $F_L$ |
|---|---|---|---|---|---|---|---|
| BFAC 2D-1 | Mar 10, 2010 | 2 global | BE | 1.0 | 0.117412 | | |
| BFAC 2D-1 | Mar 10, 2010 | 2 global | FS | 1.0 | 0.117412 | | |
| BFAC 2D-1 | Mar 10, 2010 | 2 global | CN(k) | 1.0 | 0.117412 | | |
| BFAC 2D-1 | Mar 10, 2010 | 2 global | CN | 1.0 | 0.117383 | | |
| BFAC 2D-2 | Mar 10, 2010 | 2 global | CN(k) | 1.0 | 2.31541 | | |
| BFAC 2D-2 | Mar 10, 2010 | 3 global | CN(k) | 1.0 | 2.32011 | | |
| BFAC 2D-2 | Mar 12, 2010 | 3 global | CN(k) | 1.0e-2 | 2.50288 | | |

**Code validation for FSI with ALE**

The results are summarized below:

**Program description**

| Config. | Date | Refinement | Time dis. | $k$ | $u_x(A)[\times 10^{-5}]$ | $u_y(A)[\times 10^{-4}]$ | $F_D$ | $F_L$ |
|---|---|---|---|---|---|---|---|---|
| FSI 1 | Mar 12, 2010 | 3 global | BE | 1.0 | 8.2003 | 2.2732 | | |
| FSI 1 | Apr 08, 2010 | 2 global | BE | 1.0 | 8.2258 | 2.2813 | | |
| FSI 1 | Apr 08, 2010 | 2 global | CN(k) | 0.5 | 8.2268 | 2.2813 | | |
| FSI 1 | Apr 08, 2010 | 2 global | CN | 0.5 | 8.2268 | 2.2813 | | |

### 3.2.3 Black-Scholes Equation

**General problem description**

The problem under consideration is the so called multivariate Black-Scholes equation arising from pricing European style options in finance.
To state the general form of the equation we need some nomenclature: We consider an option on $d$ risky assets with *maturity* $T > 0$ and *strikeprice* $K > 0$. For the sake of simplicity we assume the *interest rate* $r > 0$ and the *volatility* of the i-th asset $\sigma_i > 0$, $1 \leq i \leq d$, to be constant. Besides, we assume the matrix $\rho = (\rho_{ij})$ of the *correlation factors* $\rho_{ij}$ with $-1 \leq \rho_{ij} \leq 1$ for $1 \leq i, j \leq d$, to be positive definite. Of course $\rho$ is symmetric with $\rho_{ii} = 1$.
With $(t, x) \in I = (0, T] \times \mathbb{R}^d_+$ denoting the prices of the underlying assets at time $t$, the problem of determining the fair price $u$ of such an option is (after a time reversal) given by the following equation:

$$\partial_t u - \frac{1}{2} \sum_{i,j=1}^{d} \sigma_i \sigma_j \rho_{ij} x_i x_j \partial_{x_i} \partial_{x_j} u - r \sum_{i=1}^{d} x_i \partial_{x_i} u + ru = 0 \qquad \text{in } (0, T] \times \mathbb{R}^d_+, \tag{3.9a}$$

$$u(0) = u_0 \qquad \text{in } \mathbb{R}^d_+. \tag{3.9b}$$

The initial condition $u_0 \in C^0\left(\mathbb{R}^d_+\right)$ (i.e. the *payoff*) is given depending of the type of the option. For example

$$u_0 := \begin{cases} \max(\sum_{i=1}^{d} \lambda_i x_i - K, 0), & u \text{ is a } \textit{Call}, \\ \max(K - \sum_{i=1}^{d} \lambda_i x_i, 0), & u \text{ is a } \textit{Put}, \end{cases} \tag{3.10}$$

for a plain vanilla European option on a basket of assets containing a share of $0 < \lambda_i \leq 1$ of the $i$-th asset. For the computation, we truncate the domain, i.e. we choose $\overline{x} \in \mathbb{R}^d_+$ and consider the computational domain $\Omega := (x_1, \overline{x}_1) \times \cdots \times (x_d, \overline{x}_d)$. On the new part of the boundary $\Gamma$ with $\Gamma := \{x \in \partial\Omega | \exists_{1 \leq i \leq d} x_i = \overline{x}_i\}$) we impose asymptotic values as dirichlet conditions. For a put, we take $u|_\Gamma = 0$. We emphasize that no boundary conditions will be imposed on $\partial\Omega \setminus \Gamma$.
In this particular example we examine the case of two uncorrelated stocks (with $\lambda_1 = \lambda_2 = \frac{1}{2}$) and the following parameters:

| | 2d-Put) |
|---|---|
| actual asset value $x_0$ | (25,25) |
| strikeprice $K$ | 25 |
| maturity date $T$ | 1 |
| volatility $\sigma$ | $(\frac{1}{2}, \frac{3}{10})$ |
| cutoff $\overline{x}$ | $(100, 100)$ |
| interest rate $r$ | 0,05 |
| option value $u(T, x_0)$ | ca. 2,269172389 |

**Program description**

### 3.2.4 Heat Equation in 1D

**General problem description**

In this example we consider one of the prototypical instationary equations, the parabolic heat equation

$$\begin{aligned} \partial_t u(t,x) - \Delta u(t,x) &= f(t,x), \\ u(t,x)|_{\partial\Omega} &= g(t,x), \\ u(0,x) &= u_0(x) \end{aligned}$$

with unknown solution $u : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^d$. In our example, we consider the simplest case $d = 1$, where the Laplacian $\Delta$ reduces to $\partial_x^2$. The computational domain is $\Omega \times I = [0,1] \times [0,1]$. For further simplification, we choose the righthand side as $f = 0$ as well as homogeneous Dirichlet boundary conditions ($g = 0$). The initial condition is given by $u_0(x) = \min(x, 1-x)$.

**Program description**

There are few new things compared to the first two nonstationary examples. This is the first time we solve an equation in one spatial dimension. In most cases, the dimension dependence is covered by the `LOCALDOPEDIM` and `LOCALDEALDIM` variables (which are defined at the beginning of the *main.cc* file), but there might be some places in the code (especially your own code) where a concrete dimension number is given to an object. There you have to replace it manually. Do not forget to insert the correct dimension in the *Makefile*!
The most important feature of this example is the serial application of several time-stepping schemes. At the moment, the following schemes are available (see also example 3.2.1):

1. Forward Euler scheme (FE)
2. Backward Euler scheme (BE)
3. Crank-Nicolson scheme (CN)
4. shifted Crank-Nicolson scheme (sCN)
5. Fractional-Step-$\theta$ scheme (FS)

All these time-stepping methods are applied in the current example in order to check them and to compare their characteristics. To keep the computing time acceptable, we choose a one dimensional example.
One more innovation is the output format. We want to represent the output at single timepoints as a function graph on the space interval $[0,1]$; this can be done using *GNUPLOT*, for example, so instead of *.vtk* files as in all former examples, we now write out *.gpl* files.

### 3.2.5 Heat Equation in 2D with nonlinearity

**General problem description**

This example differs only slightly from the previous one. Again, we consider the heat equation, this time with an additional nonlinear term

$$\partial_t u(t,x,y) - \Delta u(t,x,y) + u(t,x,y)^2 = f(t,x,y),$$

but now in two space dimensions and with known solution

$$u(t,x,y) = e^{t-t^2}\sin(x)\sin(y).$$

The computational domain is $\Omega \times I = [0,\pi]^2 \times [0,1]$. From the known solution, we can compute the appropriate data

$$\begin{aligned} f(t,x,y) &= (3-2t)e^{t-t^2}\sin(x)\sin(y) + e^{(t-t^2)^2}\sin^2(x)\sin^2(y), \\ u_0(x,y) &= \sin(x)\sin(y). \end{aligned}$$

Furthermore, we have to prescribe homogeneous Dirichlet boundary conditions.

**Program description**

The new feature of this example is the nonhomogeneous righthand side. In examples 3.1.7 and 3.1.8, we regarded stationary problems with nonhomogeneous righthand sides, but up to now, we never involved the time variable into the nonhomogeneity. To do this, DOpE yields a `SetTime()` function which has to be applied in the *localpde.h* file as well as at the place where the `RightHandSideFunction` class is declared (here the *myfunctions.h* file.

# 4 Examples with Optimization

## 4.1 Subject to a Stationary PDE

### 4.1.1 Distributed control with a linear elliptic PDE

This example solves the distributed minimization problem

$$\begin{aligned} \min J(q,u) &= \frac{1}{2}\|u-u^d\|^2 + \frac{\alpha}{2}\|q\|^2 \\ \text{s.t.}(\nabla u, \nabla\phi) &= (q+f,\phi) \;\; \forall\, \phi \in H_0^1(\Omega) \end{aligned}$$

on the domain $\Omega = [0,1]^2$, and the data is chosen as follows:

$$\begin{aligned} f &= \left(20\pi^2 \sin(4\pi x) - \frac{1}{\alpha}\sin(\pi x)\right)\sin(2\pi y) \\ u^d &= \left(5\pi^2 \sin(\pi x) + \sin(4\pi x)\right)\sin(2\pi y) \end{aligned}$$

and $\alpha = 10^{-3}$. Hence its solution is given by:

$$\begin{aligned} \overline{q} &= \frac{1}{\alpha}\sin(\pi x)\sin(2\pi y) \\ \overline{u} &= \sin(4\pi x)\sin(2\pi y) \end{aligned}$$

In addition the following functionals are evaluated:

$$\text{MidPoint: } u(0.5;0.5)$$

$$\text{MeanValue: } \int_\Omega u$$

### 4.1.2 Parameter control with a linear elliptic PDE

This example solves the distributed minimization problem

$$\begin{aligned} \min J(q,u) &= \frac{1}{2}\|u-u^d\|^2 + \frac{\alpha}{2}\|q\|^2 \\ \text{s.t.}(\nabla u, \nabla\phi) &= (f(q),\phi) \;\; \forall\, \phi \in H_0^1(\Omega;\mathbb{R}^2) \end{aligned}$$

on the domain $\Omega = [0,1]^2$, and the data is chosen as follows:

$$\begin{aligned} f(q) = q_0 &\begin{pmatrix} 2\pi^2 \sin(\pi x)\sin(\pi y) \\ 0 \end{pmatrix} \\ + q_1 &\begin{pmatrix} 5\pi^2 \sin(\pi x)\sin(2\pi y) \\ 0 \end{pmatrix} \\ + q_2 &\begin{pmatrix} 0 \\ 8\pi^2 \sin(2\pi x)\sin(2\pi y) \end{pmatrix} \end{aligned}$$

with $\alpha = 0$ and $u^d$ such that the solution is given by:

$$\begin{aligned} \overline{q} &= \big(1; 0.5; 1\big) \\ \overline{u} &= \begin{pmatrix} \sin(\pi x)(\sin(\pi y) + 0.5\sin(2\pi y)) \\ \sin(2\pi x)\sin(2\pi y) \end{pmatrix} \end{aligned}$$

### 4.1.3 Parameter control with a nonlinear PDE from fluid dynamcis

In this example, we solve a optrimization problems from fluid dynamcis. The configuration is similar to the fluid optimization problem proposed by Roland Becker in 2000 “Mesh adaption for stationary flow control”.

We describe the underlying equations later...

### 4.1.4 Control in the dirichlet boundary values

This example solves the minimization problem

$$\begin{aligned} \min J(q,u) &= \frac{1}{2}\|u-u^d\|^2 + \frac{\alpha}{2}\|q\|^2 \\ \text{s.t. } (\nabla u, \nabla \phi) &= (f,\phi) \;\; \forall\, \phi \in H_0^1(\Omega;\mathbb{R}^2) \end{aligned}$$

on the domain $\Omega = [0,1]^2$. In addition we set the dirichlet data of the state on the boundary as follows

$$u_0(0,y) = q_0, \quad u_0(1,y) = q_1, \quad u_0(x,0) = q_2, \quad u_0(x,1) = q_3,$$
$$u_1 = q_4^3.$$

The data is chosen as follows:

$$f = \begin{pmatrix} 20\pi^2 \sin(\pi x)\sin(\pi y) \\ 1 \end{pmatrix}$$
$$u^d = \begin{pmatrix} \sin(\pi x)\sin(\pi y) * x \\ x \end{pmatrix}$$

with $\alpha = 10$.

### 4.1.5 Distributed Control with Different Meshes for Control and State

This example solves the distributed minimization problem

$$\begin{aligned} \min J(q,u) &= \frac{1}{2}\|u-u^d\|^2 + \frac{\alpha}{2}\|q\|^2 \\ \text{s.t.} (\nabla u, \nabla \phi) &= (q+f,\phi) \;\; \forall\, \phi \in H_0^1(\Omega) \end{aligned}$$

on the domain $\Omega = [0,1]^2$, and the data is chosen as follows:

$$\begin{aligned} f &= \left(20\pi^2 \sin(4\pi x) - \frac{1}{\alpha}\sin(\pi x)\right)\sin(2\pi y) \\ u^d &= \left(5\pi^2 \sin(\pi x) + \sin(4\pi x)\right)\sin(2\pi y) \end{aligned}$$

and $\alpha = 10^{-3}$. Hence its solution is given by:

$$\begin{aligned} \overline{q} &= \frac{1}{\alpha}\sin(\pi x)\sin(2\pi y) \\ \overline{u} &= \sin(4\pi x)\sin(2\pi y) \end{aligned}$$

In addition the following functionals are evaluated:

$$\text{MidPoint: } u_h(0.5;0.5)$$

$$\text{L1-Value: } \int_\Omega |u_h| \text{QError: } \int_\Omega |q_h - \overline{q}|^2 \text{UError: } \int_\Omega |u_h - \overline{u}|^2$$

The important new feature is that we can now use two different meshes for control and state variable. This is tested first for globally refined meshes, and then for locally refined meshes with different refinements for the control and state variable.

### 4.1.6 Compliance Minimization of a variable Thickness MBB-Beam

This example implements the minimum compliance problem for the thickness optimization of an MBB-Beam. Using the MMA-Method of K. Svanberg together with an augmented Lagrangian approach for the subproblems following M. Stingl.

### 4.1.7 Distributed control with a linear elliptic PDE using SNOPT

This example solves the distributed minimization problem

$$\begin{aligned} \min J(q,u) &= \frac{1}{2}\|u-u^d\|^2 + \frac{\alpha}{2}\|q\|^2 \\ \text{s.t.} (\nabla u, \nabla \phi) &= (q+f,\phi) \;\; \forall\, \phi \in H_0^1(\Omega) \\ \text{s.t.} -500 \leq q &\leq 500 \text{ a.e. in } \Omega \end{aligned}$$

on the domain $\Omega = [0,1]^2$, and the data is chosen as follows:

$$\begin{aligned} f &= \left(20\pi^2 \sin(4\pi x) - \frac{1}{\alpha}\sin(\pi x)\right)\sin(2\pi y) \\ u^d &= \left(5\pi^2 \sin(\pi x) + \sin(4\pi x)\right)\sin(2\pi y) \end{aligned}$$

and $\alpha = 10^{-3}$.

In addition the following functionals are evaluated:

$$\text{MidPoint: } u(0.5;0.5)$$

$$\text{MeanValue: } \int_\Omega u$$

The Problem is similar to that of `OPT/StatPDE/Example1` except for the box control constraints. Another new feature is the use of the commercial optimization library SNOPT. In order to use this library you need to install SNOPT on your computer and then generate a symlink to the snopt directory (where you have the libs and the header files) in the `DOpE/ThirdPartyLibs` directory named `snopt`, i.e., you should have the file `DOpE/ThirdPartyLibs/snopt` pointing to the snopt directory. If you have not done this you can compile the example but when running the example you will only get an error message like

```
Warning:  During execution of ‘Reduced_SnoptAlgorithm::Solve‘
the following Problem occurred!
To use this algorithm you need to have SNOPT installed!
To use this set the WITH_SNOP CompilerFlag.
```

### 4.1.8 Topology optimization of an MBB-Beam using SNOPT

This example implements the topology optimization of an MBB-Beam given in `OPT/StatPDE/Example6` using the SIMP method.

The solution is computed using the commercial optimization library SNOPT as in `OPT/StatPDE/Example7`. This Example demonstrate how global constraints on the control variable can be included into the optimization call.

### 4.1.9 Parameter control with a non-linear PDE from FSI dynamics

# 5 Testing examples

The DOpE testsuite consists of regression tests. They are run to compare the output to previous outputs. This is useful (necessary) after changing programming code anywhere in the library. If a test succeeds, everything is fine in the library. If not, you should not check in your code into DOpE. Please make sure what is going wrong and WHY!

Every command is computed via a Makefile. In the basic example directory

## 5.1 Where can I find the tests

In each example directory you find a subdirectory 'Test'. Herein, you find the parameter files for meshes (*.inp) and a param file (test.prm). Moreover, the executable is denoted by 'test.sh'. Please make sure, that the

```
set never_write_list
```

contains every possible output

```
Gradient;Hessian;Tangent;Residual;Update;Control;State
```

That means, no solution files are written to the output. Recall, that we are just interested in terminal output that is of course sufficient to verify the things.

Hence, the results directory should be empty

```
set results_dir   = ./
```

The rest in the param file must be identically the same as in the dope.prm file in the parent directory.

## 5.2 How to start testing?

You start testing by typing

```
> ./test.sh Store
```

in the terminal.

After the run, you have to call

```
> ./test.sh Test
```

to compare your stored output. Of course, there should be no differences.

The useful point is now the following. After implemention of new pieces of code in the DOpE library or in the examples, you can run

```
> ./test.sh Test
```

Hereby, you compare your 'new' output with the previous stored output.
Attention: After changes you should NOT run again

```
> ./test.sh Store
```

In that case, you overwrite your previous output.

# Index