DOpElib: The Differential Equations and Optimization Environment

Christian Goll[1] and Thomas Wick[2] and Winnifried Wollner[3]

[1] Universität Heidelberg,
[2] The University of Texas at Austin,
[3] Universität Hamburg

July 14, 2014
PDESoft 2014
Heidelberg

Overview

Overview

Motivation for the Software Library

$$\begin{aligned} &\text{Minimize } J(q,u) \\ &\text{subject to } \mathcal{A}(q,u) = 0 \end{aligned}$$

Solve for first order necessary conditions, e.g., find (q,u,z) s.t.

$$\begin{aligned} \mathcal{A}(q,u) &= 0 \\ \mathcal{A}'_u(q,u)^* z &= J'_u(q,u) \\ \mathcal{A}'_q(q,u) &= J'_q(q,u) \end{aligned}$$

possibly more additional constraints.

Solving the Optimization Problem

Use Newton's-method for the whole KKT system, or introduce solution operator S by

$$\mathcal{A}(q, Sq) = 0$$

then solve

$$\text{Minimize } j(q) := J(q, Su)$$

by Newton's-method applied to $j'(q) = 0$.

Motivation for the Software Library - Examples

$$\text{Minimize } \frac{1}{2}\sum_i \|v(x_i) - v_i^d\|^2 + \frac{1}{2}\alpha\|q\|^2$$

$$\text{s.t. } \begin{cases} -\nu\Delta v + v\cdot\nabla v + \nabla p = 0 \ \text{ in } \Omega \\ \nabla\cdot v = 0 \ \text{ in } \Omega, \\ \text{Boundary conditions depending on } q, \end{cases}$$

$$\text{Minimize } f(u)$$

$$\text{s.t. } \begin{cases} (E\epsilon(u), \epsilon(\phi)) = f(\phi) \ \ \forall\, \phi \in H_D^1(\Omega, \mathbb{R}^d) \\ 0 \le E, \quad \int_\Omega \operatorname{tr}(E) \le V_{\max}, \quad 0 \le \operatorname{tr}(E) \le b. \end{cases}$$

Why not use what we have?

Some given FE-Toolkit (i.e., deal.II, dune, fenics, Gascoigne ...)

- Efficient optimization routines need to know a lot of structure, i.e., derivatives need to be scaled correctly.
- Calculation of derivatives of PDE solutions w.r.t input straightforward, but not always easy to implement.

Some given OPT-Toolkit (i.e., ipopt, snopt ...)

- PDEs usually require special solvers, i.e., $Ax = b$ may need special solvers usually available in PDE software.
- OPT-Toolkit needs to know topology (not all norms are equal!)

A similar idea is followed by the coupling of Gascoigne/RoDoBo libraries.

Motivation for the Software Library - Goals

- Solving a PDE is a 'lowlevel' problem. (Its not easy!)
  - User should only implement what is necessary.
    Write equations, not loops over cells!
    $\rightarrow$ PDE becomes a template argument for other objects like solvers.
- On the other hand they are all a little bit different ...
  - Switching between finite elements should be easy.
  - Changing timesteping schemes should be possible.
- Good optimization routines are hard to implement, and should be reusable. But, sometimes you may want to switch them.
- Need to be able to access/change all modules.

Work on your research area, don't bother with the rest!

Overview

Stationary Model

$$a(u)(\phi) = (f, \phi) \quad \forall \phi \in V.$$

For example stationary incompressible Navier-Stokes equations, i.e., $u = (v, p)$ and $\phi = (\phi_u, \phi_p)$ and

$$a(u)(\phi) = \nu(\nabla v, \nabla \phi_u) + (v \nabla v, \phi_u) + (p, \nabla \cdot \phi_u) + (\nabla \cdot v, \phi_p)$$

Not only calculate u, or u_h resp., but some functionals $J(u_h)$ (in points, over lines, or on domain).

Setup

Use FE-kit for discretization. (Here `deal.II`)

- subdivision $\mathcal{T}_h$
- initialize FEs, e.g., $\mathcal{Q}_2/\mathcal{Q}_1$
- remove constraints, e.g., hanging nodes, ...

$\rightarrow$ `SpaceTimeHandler`

```
template<typename TRIANGULATION, typename FE>
    void SpaceTimeHandler(TRIANGULATION& triag, FE& fe)
```

Setup

Solve the discrete equation; here Newton, i.e., we need to calculate the residual

$$a(u_h)(\phi_h^i) - (f, \phi_h^i) = \sum_{K \in \mathcal{T}_h} \int_K \nu \nabla u_h \cdot \nabla \phi_h^i \, dx + \dots$$

$\rightarrow$ `Integrator` to integrate user defined object `pde`

```
template<typename PROBLEM, typename VECTOR>
  void Integrator
       ::ComputeNonlinearResidual(PROBLEM& pde,
                                  VECTOR & residual);
```

Needs access to `SpaceTimeHandler`. Similar:

```
template<typename PROBLEM, typename MATRIX>
  void Integrator
         ::ComputeMatrix(PROBLEM& pde,
                         MATRIX & system_matrix);
```

Setup

Now the `NewtonSolver` works:

```
template<typename LINSOLVE>
 template<typename PROBLEM, typename VECTOR>
    void NewtonSolver<LINSOLVE>
            ::NonlinearSolve(PROBLEM& pde,
                             VECTOR & solution);
```

with some appropriate linear solver `LINSOLVE`.

Setup

Bundle solution and evaluation in `StatPDEProblem`. This gives the method

```
template<class NEWTON>
    void StatPDEProblem<NEWTON>
            ::ComputeReducedFunctionals();
```

Additionally, we use a wrapper for all user given data, i.e., the template argument `PROBLEM`.

$\rightarrow$ `PDEProblemContainer`

(Important for time-dependent problems and optimization)

User Defined Data

Discretization Specification of discretization issues such as finite elements, quadrature rules, and triangulation.

Solvers Choice of a (non-)linear solver.

Problem-data Definition of the weak form, the corresponding matrices, the functionals J, as well as the boundary conditions.

Additional features ParameterReader to read in the run-time parameters such as material coefficients or adjustments for the solvers like tolerance for Newton's method.

main.cc

```
ParameterReader pr;
LocalPDE pde;
LocalFunctional functional;
SpaceTimeHandler dof_handler(triangulation,
                             fe);

PDEProblemContainer<LocalPDE>
            prob_container(pde, dof_handler);
prob_container.AddFunctional(&functional);
prob_container.SetBoundaryFunctionalColors(1);
DirichletData dd;
prob_container.SetDirichletBoundaryColors(0,
                                          comp_mask,
                                          dd);
```

main.cc

```
typedef StatPDEProblem<NEWTON> PDE;
typedef DirectLinearSolver<MATRIX, VECTOR> LINSOLVE;
typedef NewtonSolver<LINSOLVE> NEWTON;

PDE pde(prob_container, pr,
                    quad, face_quad);


pde.ComputeReducedFunctionals();
```

pde.h

```
template <typename CellDataContainer, typename VECTOR>
 void CellEquation(const CellDataContainer &cdc,
 VECTOR &local_vector)
 {
    const auto& fe_values = cdc.GetFEValuesState();
    unsigned int dofs_per_cell = cdc.GetNDoFsPerCell();
    unsigned int n_q_points = cdc.GetNQPoints();
    cdc.GetValuesState("state", uvalues);
    cdc.GetGradsState("state", ugrads);
```

pde.h

```
    for(unsigned int q_point = 0;
        q_point < n_q_points;
        q_point++)
    {
      //initialize phi_i_grads_v etc.
      Tensor<2,2> vgrads;
      vgrads[0]= ugrads[q_point][0];
      vgrads[1]= ugrads[q_point][1];
      for (unsigned int i=0;i<dofs_per_cell;i++)
        local_vector(i) +=
            ( _nu * vgrads * phi_i_grads_v
              + //all the other terms
            ) * fe_values.JxW(q_point);
    }
}
```

DWR for Navier-Stokes

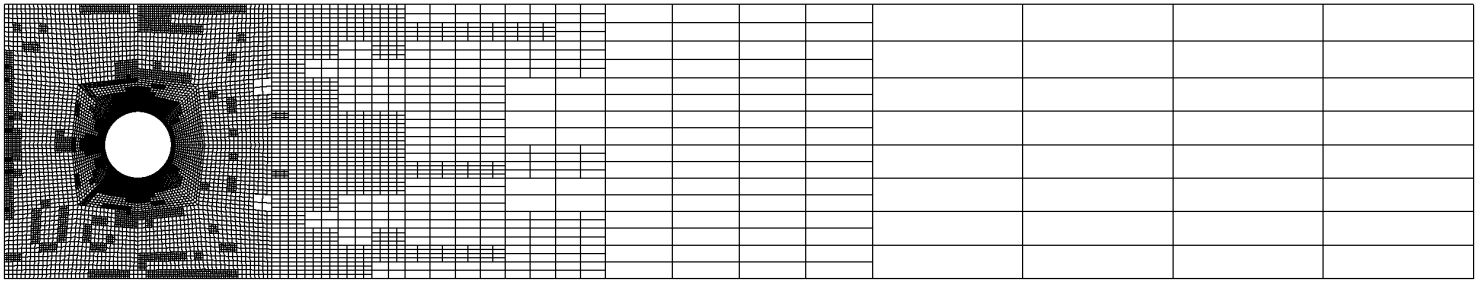

Figure : Locally adapted grid after eight refinement steps.

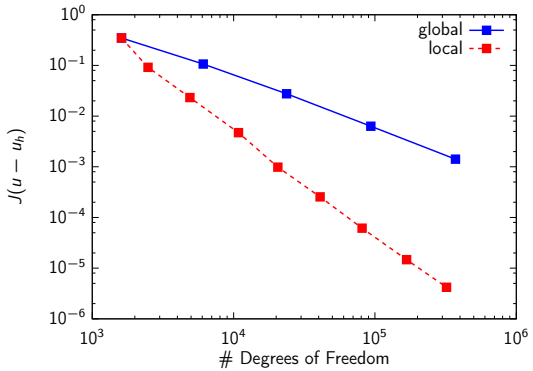

Figure : Comparison of global and local mesh refinement with respect to the error in the drag-functional J.

3D Example with Local Mesh Refinement

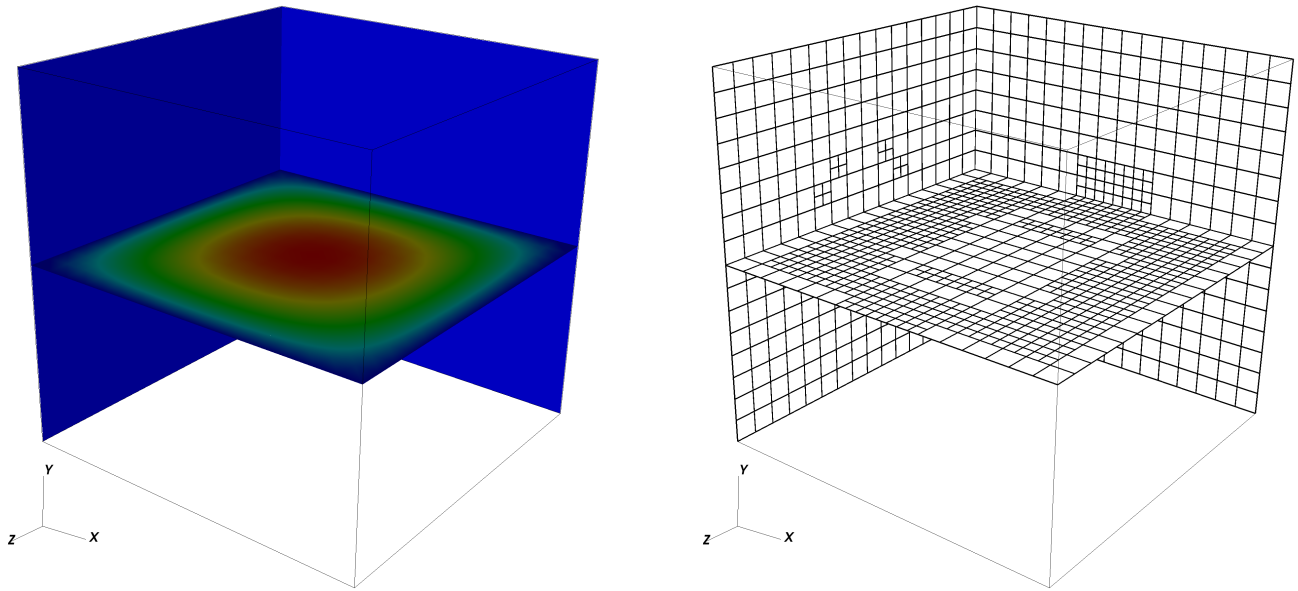

Figure : Solution and local mesh refinement.

Overview

Extension of the Model

$$\int_I \overline{a}(u)(\Phi)\,dt = \int_I (f,\Phi)\,dt \quad \forall \Phi \in X,$$

For our example:

$$\int_I \left[(\partial_t v,\phi) + \nu(\nabla v,\nabla\phi) + (v\cdot\nabla v,\phi) - (p,\nabla\cdot\phi) + (\nabla\cdot v,\chi)\right] dt$$
$$+ (v(0)-v^0,\phi(0)) = \int_I (f,\phi)\,dt$$

for all $\Phi := (\phi,\chi)$.
The only difference:

$$\int_I (\partial_t v,\phi)\,dt$$

Setup

Modified constructor for the `SpaceTimeHandler`

```
template<typename TIMES, typename TRIANGULATION,
         typename FE>
 void SpaceTimeHandler(TIMES& times,
                       TRIANGULATION& triag,
                       FE& fe);
```

Need discretization of the term ∂_t.

```
template<typename PROBLEM>
    class TimeStepProblem(Problem & P);
```

- Forward-/Backward-Euler
- Crank-Nicolson scheme
- Shifted Crank-Nicolson scheme (to deal with singular initial data)
- Fractional-Step-θ scheme

The Time Derivative

$$(\partial_t v, \phi) + (\nabla v, \nabla \phi) + (v \cdot \nabla v, \phi) \\ - (p, \nabla \cdot \phi) + (\nabla \cdot v, \chi) = (f, \phi),$$

Discretize with θ scheme:

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

`scale` $= \pm k\theta$ or `scale` $= \pm k(1-\theta)$ and `scale_ico` $= \pm k\theta$.

pde.h

```
void CellEquation(...)
 {
 // Usual loops over DoFs and quadrature points
   {
    local_cell_vector(i) +=
       scale *
       // terms like (\nabla v^{n+1}, \nabla\phi) etc.
     + scale_ico *
       // terms like (-p^{n+1},\nabla\cdot\phi)
   }
 }
```

Can reuse stationary code, if `scale`, `scale_ico` are properly used.

pde.h

The time-derivative, i.e., the term (v^{n+1}, ϕ):

```
void CellTimeEquation(...)
 {
 // Usual loops over DoFs and quadrature points
      local_cell_vector(i) +=  scale * (v * phi_i)
                                     * fe_values.JxW(q_point);
 }
```

Nonlinear-time derivatives can be handled by explicit implementation

```
void CellTimeEquationExplicit (...)
 {
   // implement (v^{n+1},\phi) and (v^n,\phi)
 }
```

Nonstationary FSI

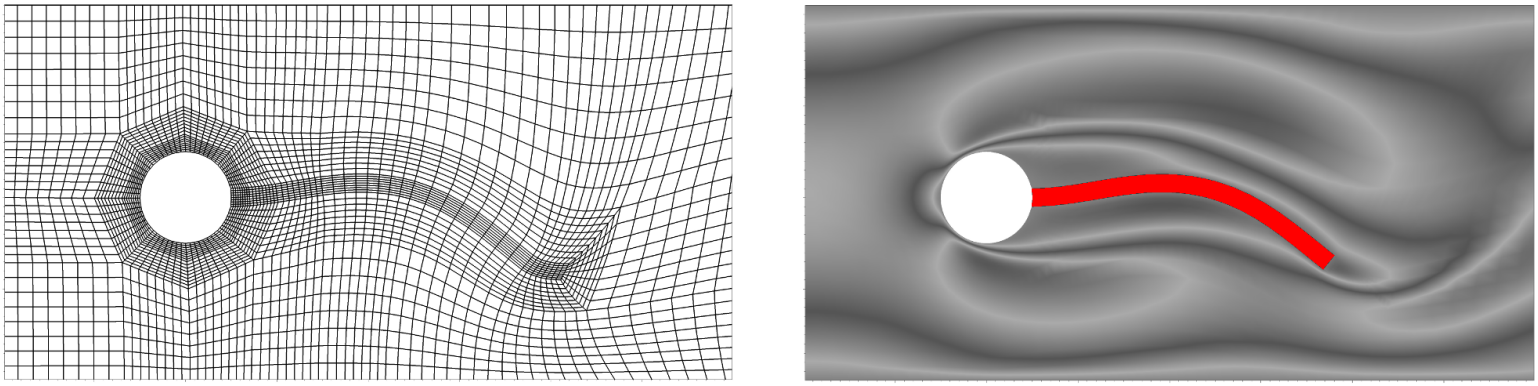

Figure : FSI 2 test case: mesh (left) and velocity profile in vertical direction (right) at time $t = 16.14s$.

Nonstationary FSI

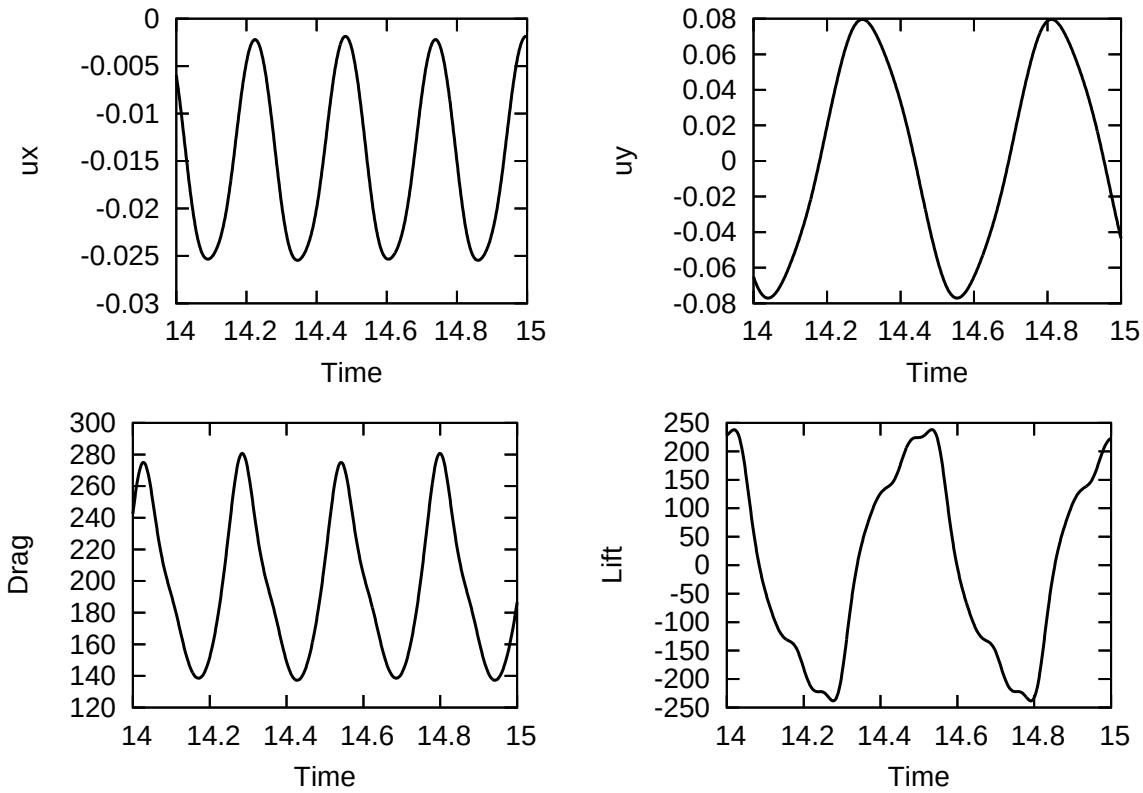

Figure : FSI 2: The deflections of the beam, $u_x(A)$ and $u_y(A)$ (in cm), and the drag F_D and the lift F_L evaluation (in $kg/m\,s^2$) are displayed versus time (in s).

Overview

Optimization Model

A prototypical PDE constrained optimization problem reads:

$$\begin{aligned} \min\ & J(q,u) \\ & \text{s.t. } a(q,u)(\phi) = 0 \quad \forall \phi \in V, \\ & q_- \leq q \leq q_+, \quad g(q,u) \leq 0, \end{aligned}$$

To keep the presentation brief we will neglect the additional bounds

$$q_- \leq q \leq q_+, \qquad g(q,u) \leq 0.$$

although they can be treated by `DOpElib` as well.
Eliminate the PDE

$$a(q,S(q))(\phi) = 0 \quad \forall \phi \in V.$$

$\rightarrow$

$$\min j(q) = J(q,S(q)).$$

Additional Requirements

Directional derivative of j in direction δq is given by

$$j_h'(q)(\delta q) = J_q'(q, S_h(q))(\delta q) + (S_h^* J_u'(q, S_h(q)), \delta q).$$

Calculate adjoint $z = S_h^* J_u'(q, S_h(q))$ via the adjoint PDE

$$a_u'(q, u)(\phi, z) = J_u'(q, u)(\phi) \quad \forall \phi \in \mathcal{X}.$$

Similar formulas for the second derivatives.

Optimization Problems

Replace `PDEContainer` with

```
template<typename PDE, typename COSTFUNCTIONAL>
  class OPTProblemContainer;
```

to give access to derivative information:

```
void CellEquation_U(...)
 {
   // implement directional derivative
   // of the CellEquation w.r.t. u
 }
```

Then we can solve the opimization problem, e.g., by

```
ReducedNewtonAlgorithm::Solve(ControlVector<VECTOR>& )
```

Compliance Minimization

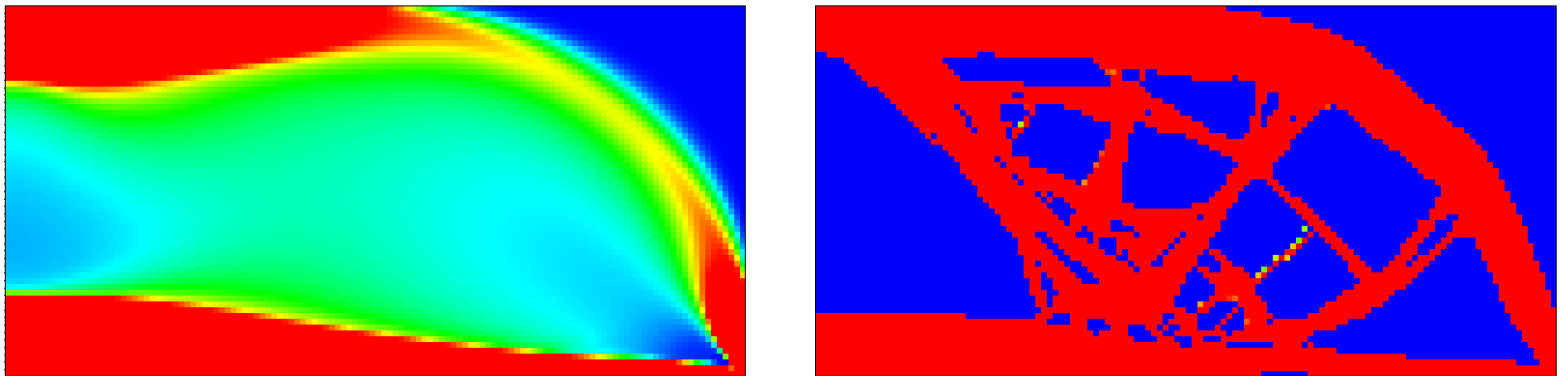

Figure : Thickness distribution for the MBB-beam minimum compliance problem using SIMP parameter $p = 1$ (left) and $p = 4$ right.

Outflow Maximization for FSI

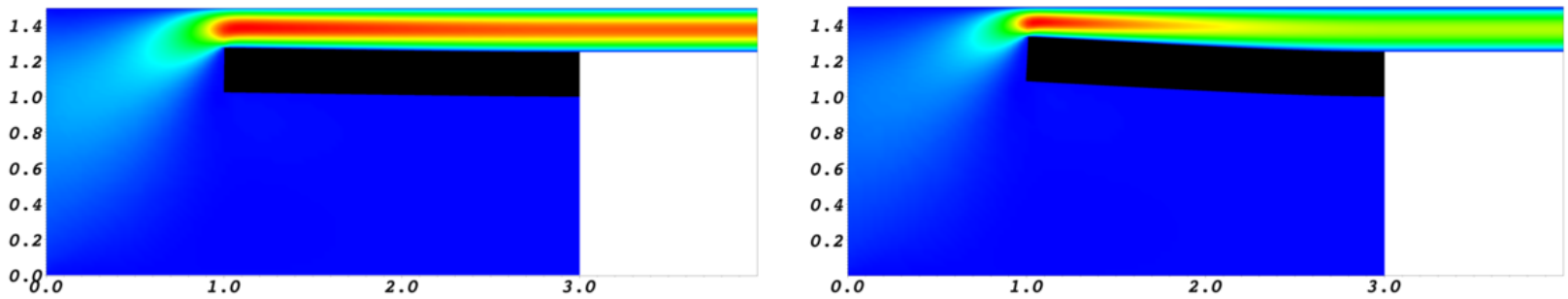

Figure : Maximization of outflow rate by Neumann control: Velocity profile for the initial control $p_{\text{in}} = 0.1$ (left) and velocity profile for the optimal control $p_{\text{in}} \approx 0.23$ (right). Specifically, the deformation of the beam, reducing the width of the channel, is seen at right.

Overview

Features

- PDE-Solvers
  - Newton-Method for PDE-problems
  - Backward and Forward Euler, CN, Shifted CN, Fractional-Step-Θ
- Optimization Routines
  - Reduced Newton Method (TR and LS)
  - Interface to SNOPT, IPOPT
- Test Cases
  - PDE-Problems in 2D and 3D
  - PDE-Optimization Problems
  - Multiphysics Problems in Non-Overlapping Domains
  - Benchmarks (code validation!!!) in Elasticity, Plasticity, Fluid Dynamics, and FSI
- Auxilliary Features
  - Transparent Use of Multiple Meshes
  - DWR and Residual Error Estimation

Thank You for Your Attention!

Differential Equations and Optimization Environment: DOpElib

http://www.dopelib.net/


C. Goll, T. Wick, W. Wollner
DOpElib: Differential Equations and Optimization Environment; A Goal Oriented Software Library for Solving PDEs and Optimization Problems with PDEs
Preprint, Universität Heidelberg (2012)