DD2365 Advanced Computation in Fluid Mechanics
Lab 3: Adaptive FEM for Navier-Stokes equations

Johan Jansson, Cem Degirmenci, Johan Hoffman

April 20, 2016

0 Jupyter-FEniCS web PDE solver environment

The address of the web Jupyter-FEniCS environment, described more in detail below, is provided via email with the ip of the cloud virtual machine and Jupyter login. To run a program in Jupyter-FEniCS, open the Python 2 notebook and select the Run command under the Cell menu. Do not forget to save your notebook regularly to your own computer, by using Download as ¿ IPython Notebook (.ipynb) under File in your notebook. You can also set up the environment at your own computer by using the command:

```
sudo docker run -t -i -p 80:8000 sputnikcem/fenics-jpy
```

and using the login name and passwords listed on the terminal window by accessing localhost from a web browser.

1 Introduction

In this lab session you will use the FEniCS [1] framework for automated solution of partial differential equations (PDE) to formulate and solve Navier-Stokes equations using finite element methods (FEM) with the purpose to investigate some of the concepts related to adaptivity and stability.

In this lab specifically you will investigate adaptive FEM for fluid flow modeled by the stationary incompressible Navier-Stokes equations [2, 3] in 2D as a preparation for adaptive FEM for 3D turbulent flow on the Beskow supercomputer in the Lab 4.

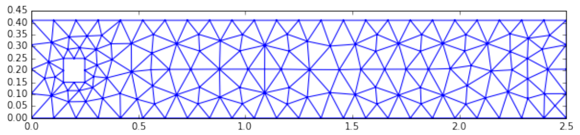
The goal of this session is to:

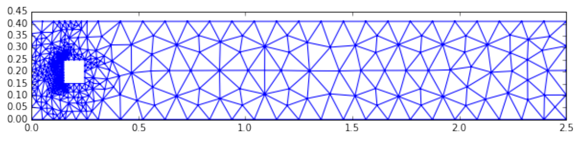
1. Become familiar and experiment with goal-oriented adaptive error control based on solving a dual/adjoint problem.

2. Become more familiar and experiment with stabilization (necessary for convection-dominated problems).

In this session we will work with the Python interface to FEniCS. We will use FEniCS version 1.6 which is installed in the Jupyter notebook (`http://jupyter.org`) Python web environment provided at the link at the top of the instructions. On the FEniCS home page [1] there is extensive documentation of the interface at both overview and detail level.

2 Exercises





The setting of this exercise is adaptive error control in FEM, which is a methodology for satisfying a tolerance on the global discretization error measured in a quantity of interest (drag/lift on an object, displacement of a region, etc.) by determining how the cells in the mesh contribute to the global error and iteratively refining those cells which have the largest contribution (or generating a new mesh in some other way to satisfy the tolerance). This gives crucial *reliability* of the method for applications since a bound on the discretization error is known, and *efficiency* since a mesh that in some sense is optimal for a given tolerance can be constructed.

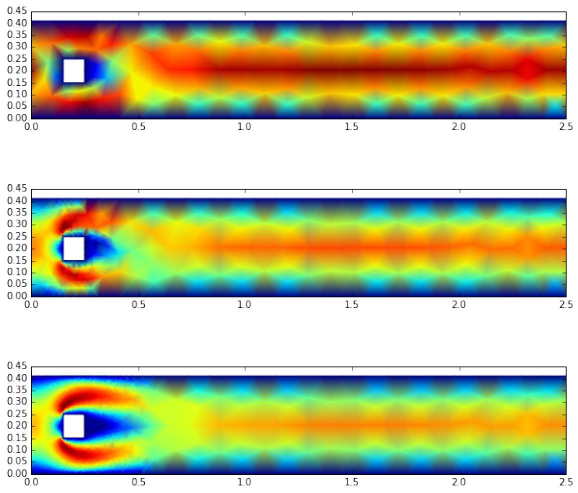


Figure 2: The velocity on the initial mesh (top), after 4 adaptive iterations (mid), and after 9 adaptive iterations (bottom) for goal functional `M1` (drag).

We will investigate the *do-nothing* [4] method for error control, a new method which is very simple in formulation and is automated in the sense that it doesn't require manual derivation for every PDE.

For a linear stationary boundary value PDE written as the weak residual $r(u, v)$ with exact solution u:

$$r(u, v) = a(u, v) - L(v) = 0, \forall v \in V \tag{1}$$

the finite element method and continuous piecewise linear (cG(1)) FEM solution U belonging to the finite element space $V_h \in V$ can be expressed as:

$$r(U, v) = a(U, v) - L(v) = 0, \forall v \in V_h \tag{2}$$

We can express the error $e = u - U$ measured in a goal functional $M(e)$ exactly using the weak residual r and an exact solution ϕ to an adjoint problem formulated below by the *error representation*:

$$M(e) = a(e, \phi) = r(U, \phi) \tag{3}$$

the do-nothing error indicator for cell K in the mesh is simply:

$$\mathcal{E}_K = r(U, \Phi)_K \tag{4}$$

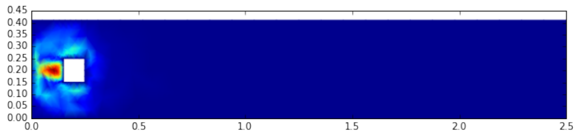


Figure 3: The adjoint velocity after 4 adaptive iterations for goal functional `M1` (drag).

with the cG(1) adjoint solution Φ defined by the following adjoint problem with the goal functional M as source:

$$a(w, \Phi) = M(w), \quad \forall w \in V_h \tag{5}$$

We will now apply the do-nothing adaptive method to the stationary incompressible Navier-Stokes equation.

2.1 Goal-oriented adaptive error control for the stationary Navier-Stokes equations

We would like to solve the system of incompressible Navier-Stokes equations, which in weak form, with weak residual r can be stated as:

$$\begin{aligned} r(\hat{u}, \hat{v}) = ((u \cdot \nabla)u + \nabla p, v) + (\nu \nabla u, \nabla v) + (\nabla \cdot u, q) = 0, \\ \hat{u} \in [V_h]^2 \times Q_h, \quad \forall \hat{v} \in [V_h]^2 \times Q_h \\ \hat{u} = (u, p) \quad \text{(Solution: velocity and pressure)} \\ \hat{v} = (v, q) \quad \text{(Test function)} \end{aligned} \tag{6}$$

with ν a diffusion parameter.

In FEniCS notation this can be written:

```
r = (inner(grad(um)*um + grad(p),v) + nu*inner(grad(um), grad(v)) + div(um)*q)*dx
```

A Galerkin-Least Squares stabilized formulation can be written:

```
# The strong residual with w the solution (w2 used for simple linerization)
def R(w, w2):

    (u, p) = (as_vector((w[0], w[1])), w[2])
    (u2, p2) = (as_vector((w2[0], w2[1])), w2[2])

    Au = grad(p) + grad(u2)*u
    Ap = div(u)

    Aui = [Au[i] for i in range(0, 2)]

    return as_vector(Aui + [Ap])

def r(W, w, wt, stab=True):

    (u, p) = (as_vector((w[0], w[1])), w[2])
    (v, q) = (as_vector((wt[0], wt[1])), wt[2])

    h = CellSize(W.mesh())
    delta = h # GLS stabilization parameter
    if(not stab):
        delta = 0.0

    F_G = (nu*inner(grad(u), grad(v)) + inner(grad(p) + grad(u)*u, v) + div(u)*q)*dx
    F_stab = delta*inner(R(w, w), R(wt, w))*dx
    F = F_G + F_stab

    return F
```

The adjoint problem can be automatically generated and solved simply like so, here a linearized adjoint problem for the general case of a non-linear weak form:

```
F = r(W, w, wt, stab=True)
a_star = adjoint(derivative(F, w))
L_star = M(v, q)

# Solve the dual problem
solve(a_star == L_star, phi, bcs)
```

and the error indicator `ei` can be expressed like so:

```
z = TestFunction(Z)
Lei = r(W, w, z*phi, stab=False)
ei = Function(Z)
ei.vector()[:] = assemble(Lei).array()
```

with Z the dG(0) finite element space of discontinuous piecewise constant functions.

The source code applying the do-nothing goal-oriented adaptive error control described above to stationary incompressible Navier-Stokes flow is available in the notebook file `labadaptive.ipynb`.

Choose either the standard exercise below, studying different data for the adaptive method, or the advanced exercise, extending the adaptive method.

2.1.1 Standard exercise

Edit the file and try different data for the adaptivity:

1. Try different goal functionals by selecting which one to return in the `M()` function:

   ```
   def M(mesh, u, p):
       n = FacetNormal(mesh)

       I = Identity(2)
       sigma = p*I - nu*epsilon(u)
       theta = Constant((1.0, 0.0))

       M1 = psimarker*p*n[0]*ds # Drag (only pressure)
       M2 = psimarker*p*n[1]*ds # Lift (only pressure)
       M3 = inner(psi, u)*dx # Mean of the velocity in a region
       M4 = psimarker*dot(dot(sigma, n), theta)*ds # Drag (full stress)
       M5 = u[0]*dx # Mean of the x-velocity in the whole domain

       return M1
   ```

   What is the effect on the final mesh? For an advanced exercise, try defining your own goal functional.

2. Experiment with setting the refinement `ratio` variable (`ratio = 0.1` refines 10% of the cells in the mesh) for different goal functionals. Compare a low ratio (10% for example) with 100% (representing uniform refinement) with regard to efficiency.

3. Experiment with the viscosity `nu`. Try for example values of 0.1, 0.01 and 0.001.

4. Try different geometries.

2.1.2 Advanced exercise

For more advanced exercises choose one of:

1. Try different stabilization methods, and no stabilization at all:
   - Artificial viscosity stabilizaton: $Ch(\nabla U, \nabla v)$ with C a stabilization parameter.
   - Unstabilized Taylor-Hood formulation, choosing a piecewise quadratic approximation for the velocity and piecewise linear for the pressure, and removing the stabilization.
2. Try to formulate a time-dependent version of the adaptive solver.

References

[1] FEniCS. Fenics project. *http://www.fenicsproject.org*, 2003.

[2] Johan Hoffman, Johan Jansson, Rodrigo Vilela de Abreu, Niyazi Cem Degirmenci, Niclas Jansson, Kaspar Müller, Murtazo Nazarov, and Jeannette Hiromi Spühler. Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid-structure interaction for deforming domains and complex geometry. *Computers and Fluids*, 2012.

[3] Johan Hoffman and Claes Johnson. *Computational Turbulent Incompressible Flow: Applied Mathematics Body and Soul Vol 4*. Springer-Verlag Publishing, 2006.

[4] Johan Jansson, Johan Hoffman, and Cem Degirmenci. Adaptive error control in finite element methods using the error representation as error indicator. Technical report, KTH, High Performance Computing and Visualization (HPCViz), 2013. QC 20131118.