PRACE Workshop, FEniCS Worksheet 1

Stockholm, December 5, 2013.

0 Download files

`http://csc.kth.se/~jjan/prace_files_ws1.zip`

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 finite element methods (FEM) for PDE and formulate and test methods for adaptive error control and stabilization.

Specifically we will investigate the General Galerkin (G2) methodology [2, 3] for adaptive stabilized FEM which is what we will use in the second lab session based on FEniCS-HPC [2].

The goal of this session is to:

1. Learn the basic interface of FEniCS: form language and function, mesh and linear algebra interfaces.
2. Become familiar and experiment with goal-oriented adaptive error control based on solving a dual/adjoint problem.
3. Become familiar and experiment with stabilization (necessary for convection-dominated problems).

In this session we will work with the Python interface to FEniCS since it's easier and faster to develop in than the C++ interface. We will use FEniCS 1.1 which is installed on the lab computers. The form language: Unified Modeling Language (UFL) that we use to formulate PDE in weak form and FE methods is the same in the Python and C++ interfaces, also in the FEniCS-HPC version developed for large-scale applications on supercomputers.

2 Exercises

2.1 FEniCS interface

We start with the simplest equation: the L_2-projection, which computes the optimal projection of a function f into a FE space V_h. To compute the L_2-projection we want to solve the equation:

$$(R(U), v) = (U, v) - (f, v) = 0, \forall v \in V_h \qquad (1)$$

We identify the terms with U and v as the **bilinear form** $a(U, v)$ and the terms with only v as the **linear form** $L(v)$.

We can thus define the representation of the equation in FEniCS as:

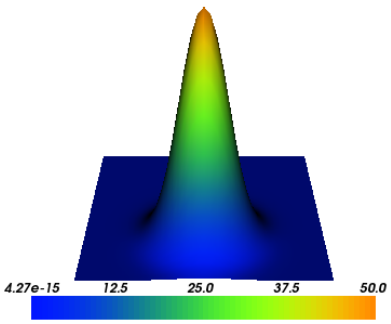

Figure 1: Plot of the L_2-projection example.

```
from dolfin import *

f = Expression("50*exp(-50*(pow(x[0] - 0.5, 2) + pow(x[1] - 0.3, 2)))")
mesh = UnitSquare(40, 40)

V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)

a = u*v*dx
L = f*v*dx
```

To solve the equation we use the compact `solve()` notation:

```
U = Function(V) # FEM solution
solve(a == L, U)
```

and we can plot with the built-in `plot()` function:

```
plot(U, interactive=True)
```

You should see something similar to the image on the right:

2.1.1 Exercise 1

The source code outlined above is available in the file `basic.py`. First try running the given program by:

```
python basic.py
```

Then edit the file and try different things, for example:

1. Add a diffusion term: $\epsilon(\nabla u, \nabla v)$ to the bilinear form.
2. Compute the L_2-norm of the solution: $\|U\|_{L_2} = \sqrt{(U, U)}$, this can be done by: `sqrt(assemble(U*U*dx))`.

2.2 Adaptive error control and stabilization

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.

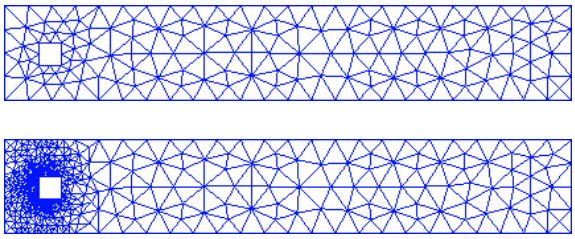

Figure 2: The initial (top) and finest (bottom) meshes for goal functional `M1` (drag).

We will investigate the *representation adaptivity* [4] method for error control, a new method which is very simple in formulation and is automated in the

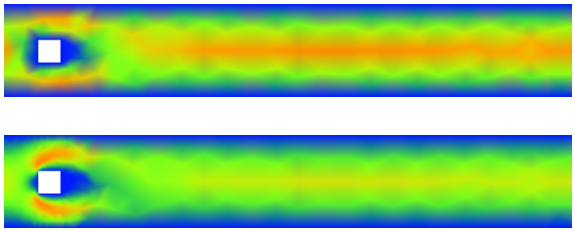

Figure 3: The velocity on the initial (top) and finest (bottom) meshes for goal functional `M1` (drag).

sense that it doesn't require manual derivation for every PDE.

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

$$r(u, v) = a(u, v) - L(v), \forall v \in V \qquad (2)$$

the error indicator for cell K in the mesh is simply:

$$\mathcal{E}_K = r(U, \Phi)_K \qquad (3)$$

with U the continuous piecewise linear (cG(1)) FEM solution and with the cG(1) dual solution Φ defined by the following dual problem with the goal functional M as source:

$$a(w, \Phi) = M(w), \quad \forall w \in V_h \qquad (4)$$

Figure 4: The dual velocity on the finest mesh for goal functional `M1` (drag).

We apply the representation adaptivity method to the stationary incompressible Navier-Stokes equation.

2.2.1 Exercise 2

The source code applying the representation adaptivity described above to stationary incompressible Navier-Stokes flow is available in the file `ns_adapt.py`. First try running the program by:

```
python ns_adapt.py
```

Edit the file and try different things, for example:

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.

2.2.2 Advanced exercise

For more advanced exercises:

1. Try a different stabilization method, or no stabilization at all, for example:
   - 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 adaptively solve a different equation, for example linear elasticity by choosing a different weak residual (including possibly the function space).

References

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

[2] Hoffman, J., Jansson, J., de Abreu, R. V., Degirmenci, N. C., Jansson, N., Müller, K., Nazarov, M., and Spühler, J. H., "Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid-structure interaction for deforming domains and complex geometry," *Computers and Fluids*, 2012.

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

[4] Jansson, J., Hoffman, J., and Degirmenci, C., "Adaptive error control in finite element methods using the error representation as error indicator," Tech. rep., KTH, High Performance Computing and Visualization (HPCViz), 2013, QC 20131118.