DD2365 Advanced Computation in Fluid Mechanics
Lab 1: FEM for Stokes equations

Johan Hoffman, Johan Jansson, Niyazi Cem Degirmenci

April 12, 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 a 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 finite element methods (FEM) and solver the resulting discrete systems. Specifically you will investigate FEM for fluid flow modeled by the incompressible Stokes equations.

The goal of this session is to:

1. Learn the basic interface of FEniCS: form language and function, mesh and solve interfaces.

2. Become familiar with the Jupyter web Python notebook interface.

3. Become familiar with different FEM methods for the the Stokes equations.

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 by following the instructions in the first section. On the FEniCS homepage [1] there is extensive documentation of the interface at both overview and detail level.

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 finite element space V_h. To compute the L_2-projection we want to solve the following equation in "weak" form: Find $u \in V_h$ such that

$$r(u, v) = (u, v) - (f, v) = 0, \quad \forall v \in V_h \tag{1}$$

where u is the unknown solution function, f is a known function, and v is a test function in a finite element test space constructed by a triangulation with cell diameter h and piecewise linear functions, with the standard notation for L_2 inner products: $(v, w) = \int_\Omega vw \, dx$.

If we replace the finite dimensional space V_h with the space V of all integrable functions (with bounded integral), the "strong" form of the equation below is equivalent:

$$R(u) = u - f = 0 \qquad (2)$$

which has the trivial solution $u = f$, with the condition that $f \in V$, i.e. f has to be integrable. In all the exercises in the lab we will use the weak form of the equations. The strong form can be derived as above if desired.

We first include the plotting interface for Jupyter:

```
%matplotlib inline
%run /home/fenics/fenics-matplotlib.py
```

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

```
from dolfin import *
import logging; logging.getLogger('UFL').setLevel(logging.ERROR)
#to get rid of annoying warnings from UFL

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

V = FunctionSpace(mesh, "CG", 1)
v = TestFunction(V)
u = Function(V) # FEM solution

r = (u*v - f*v)*dx
```

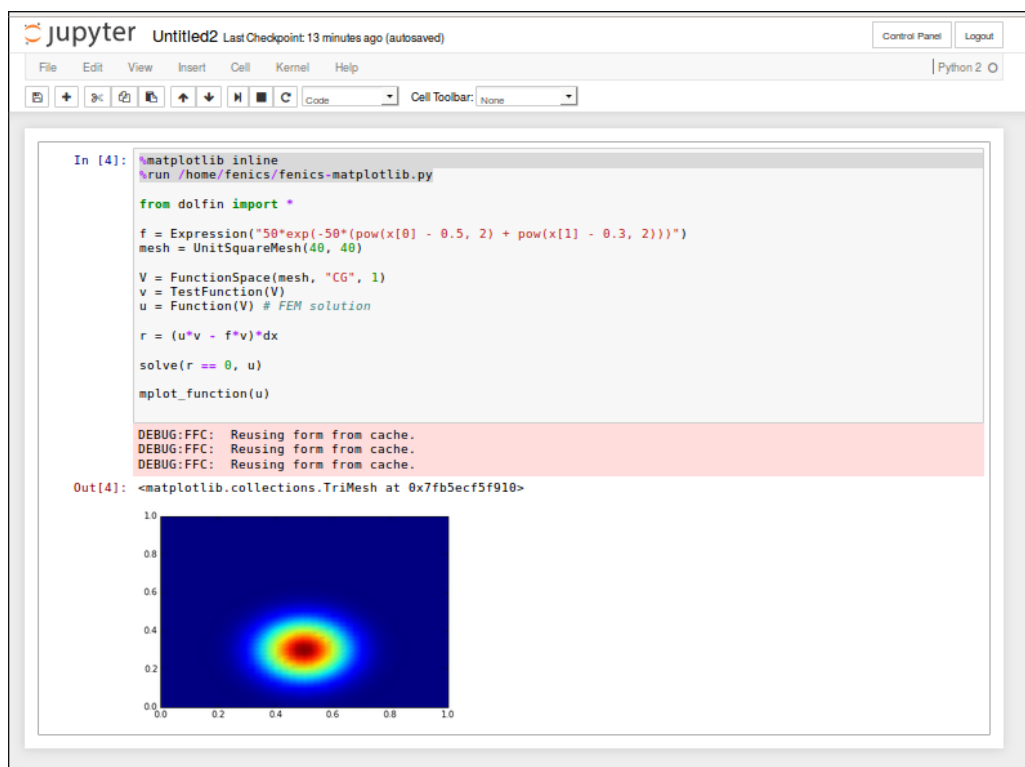To solve the equation we use the compact `solve()` notation:

```
solve(r == 0, u)
```

and we can plot in the Jupyter environment with the provided `mplot_function()` function:

```
mplot_function(u)
```

2.1.1 Screenshot

The template program and the expected output when running it should look like this:

2.1.2 Questions

Now modify the above given program:

a) Add a diffusion term to the bilinear form: $\nu(\nabla u, \nabla v)$, written in FEniCS as:

```
nu * inner(grad(u), grad(v))*dx.
```

b) Compute the max-norm of the solution, this can be done by: `u.vector().norm('linf')`.

c) The computational mesh is created with the `UnitSquareMesh` function. The mesh can be visualized by adding to your code:

```
fig2 = plt.figure()
plt.triplot(mesh2triang(mesh))
```

Try to refine and coarsen the mesh by changing the arguments to the `UnitSquareMesh` command. Use `plt.colorbar()` to also investigate results better.

d) Import the `mshr` module to your code for meshing more complex geometries. The following code for example generates a custom mesh on a rectangular domain with a circular hole:

```
XMIN = 0.; XMAX = 4.; YMIN = 0; YMAX = 1.; G = [XMIN, XMAX, YMIN, YMAX]; eps=1e-5
mesh = generate_mesh(Rectangle(Point(G[0], G[2]), Point(G[1], G[3])) - Circle(Point(.5, .5), .1), 30)
```

Try to create your own complex domains and solve the problem on that domain.

e) Add homogenous Dirichlet BC to the $x[1] = YMIN$ wall (bottom) and homogenous Neumann BC to the $x[0] = XMAX$ wall (right), in order to add the Dirichlet BC use the `DirichletBC` class and add the instanciated object as an argument to your solve function. In order to add the Neumann BC you need to modify your weak form by adding a surface integral with a ds term.

2.2 Stokes equations

The Stokes equations in weak form can be stated, using the weak residual r:

$$r(\hat{u}, \hat{v}) = (\nu \nabla u, \nabla v) - (p, \nabla \cdot v) + (\nabla \cdot u, q) = 0, \tag{3}$$
$$\hat{u} \in [V_h]^2 \times Q_h, \quad \forall \hat{v} \in [V_h]^2 \times Q_h$$
$$\hat{u} = (u, p) \quad \text{(Solution: velocity and pressure)}$$
$$\hat{v} = (v, q) \quad \text{(Test function)}$$

with ν a diffusion parameter. In FEniCS notation this can be written:

```
# FEM functions
V = VectorFunctionSpace(mesh, "CG", 1); Q = FunctionSpace(mesh, "CG", 1); W = V * Q; h = CellSize(mesh);
(v, q) = TestFunctions(W); w = Function(W); (u, p) = (as_vector((w[0], w[1])), w[2]); u0 = Function(V)

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

To fully specify the solution, we need to add known data on the boundary $\partial\Omega$ of the domain Ω, in the form of *boundary conditions*. We want to specify a known inflow velocity u_{in} profile at the left side of the domain (the inlet), zero pressure at the right edge (the outlet), and zero velocity on the rest of the boundary (a "no-slip" condition).

We will apply these boundary conditions *weakly* which means that we will add penalty terms to the weak residual, active only on the boundary, which will force the solution to the desired values if a penalty parameter is chosen large enough. We choose the penalty parameter $\gamma = 10^4$. For the inflow velocity, for example, we add the term $\gamma(u - u_{in}, v)$. If γ goes to infinity, solving the equation $r(\hat{u}, \hat{v}) = 0, \forall v \in V_h$, means that $u = u_{in}$.

We define *boundary markers* which are one on the part of the boundary we are interested in, and zero elsewhere; here `im` is the inlet marker, `om` is the outlet marker and `nm` is the no-slip marker:

```
uin = Expression (("4*(x[1]*(YMAX-x[1]))/(YMAX*YMAX)", "0."), YMAX=YMAX) # Inflow velocity
om = Expression("x[0] > XMAX - eps ? 1. : 0.", XMAX=XMAX, eps=eps) # Mark regions for boundary conditions
im = Expression("x[0] < XMIN + eps ? 1. : 0.", XMIN=XMIN, eps=eps)
nm = Expression("x[0] > XMIN + eps && x[0] < XMAX - eps ? 1. : 0.", XMIN=XMIN, XMAX=XMAX, eps=eps)

r = (nu*inner(grad(u), grad(v)) - p*div(v) + div(u)*q)*dx + \
    gamma*(om*p*q + im*inner(u - uin, v) + nm*inner(u, v))*ds
```

2.2.1   Questions

a) Plot the pressure and the velocity magnitude of the computed solution.

b) Change the order of approximation order of the finite element spaces V_h and Q_h, what happens to the computed solution? How does it effect the running time?

c) Add a stabilization term to the method.

d) Experiment with the computational domain; how does it change the solution?

References

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