DD2365 Advanced Computation in Fluid Mechanics
Lab 2: FEM for Navier-Stokes equations

Johan Hoffman

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 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 finite element methods (FEM) for PDE.

Specifically you will investigate FEM for fluid flow modeled by the incompressible Navier-Stokes equations.

The goal of this session is to:

1. Learn different time stepping methods for Navier-Stokes equations.
2. Simulate chemical reactions in a flow field computed from the Navier-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 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

2.1 Timestepping the incompressible Navier-Stokes equations

Now we will study two different timestepping methods for Navier-Stokes equations: implicit Euler (first order accurate) and the Midpoint method (second-order accurate, in principle the same method as the Trapezoid method). The timestepping method is separated from the spatial discretization, which is done by a finite element discretisation.

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

$$\begin{aligned} r(\hat{u}, \hat{v}) = (\dot{u} + (u \cdot \nabla)u + \nabla p, v) + (\nu \nabla u, \nabla v) + (\nabla \cdot u, q) = 0, \qquad (1) \\ \hat{u} \in [V_h]^2 \times Q_h, \quad \forall \hat{v} \in [V_h]^2 \times Q_h \\ \hat{u} = (u, p) \quad \text{(Solution: velocity and pressure)} \\ \hat{v} = (v, q) \quad \text{(Test function)} \end{aligned}$$

with ν a diffusion parameter.

In FEniCS notation this can be written:

```
r = (inner((u − u0)/k + grad(p) + grad(um)*um, v) + nu*inner(grad(um), grad(v)) + div(um)*q)*dx
```

with k the time step, $u0$ the velocity from the previous time step, and $um = \frac{u+u0}{2}$. Here we have applied the Midpoint timestepping method, which is why we evaluate the solution at the mean value in the timestep $\frac{u+u0}{2}$.

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 edge 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 1 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:

```
r = (inner((u − u0)/k + grad(p) + grad(um)*um, v) + nu*inner(grad(um), grad(v))
  + div(um)*q)*dx +  gamma*(om*p*q + im*inner(u − uin, v) + nm*inner(u, v))*ds
```

To be able to guarantee *stability* of the solution, i.e. that the solution and its derivatives are bounded by the given data, we use a stabilized method, which adds stabilization terms with a stabilization parameter d:

```
r = (inner((u − u0)/k + grad(p) + grad(um)*um, v) + nu*inner(grad(um), grad(v))
  + div(um)*q)*dx +   gamma*(om*p*q + im*inner(u − uin, v) + nm*inner(u, v))*ds
  + d*(inner(grad(p) + grad(um)*um, grad(q) + grad(um)*v)
  + inner(div(um), div(v)))*dx # Stabilization
```

again in FEniCS we can solve the equation for one timestep simply by:

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

Finally we carry out all the timesteps in the time interval $I = [0, T]$ with a timestepping loop:

```
t, T = 0., 10. # Time interval
while t < T: # Time−stepping loop

   ... # Solve the Navier−Stokes PDE (one timestep)

   t += k; u0 = project(u, V); # Shift to next timestep
```

Plotting is implemented in the `plot_compact()` utility function, using the *Matplotlib* interface.

```
def plot_compact(u, t, stepcounter): # Compact plot utility function
    if stepcounter % 5 == 0:
        uEuclidnorm = project(sqrt(inner(u, u)), Q); ax.cla(); fig = plt.gcf(); fig.set_size_inches(16, 2)
        plt.subplot(1, 2, 1); mplot_function(uEuclidnorm); plt.title("Velocity") # Plot norm of velocity
        if t == 0.: plt.colorbar(); plt.axis(G)
        plt.subplot(1, 2, 2);
        if t == 0.: plt.triplot(mesh2triang(mesh)); plt.title("Mesh") # Plot mesh
        plt.suptitle("Lab 4 ICNS - t: %f" % (t));  plt.tight_layout(); clear_output(wait=True); display(pl)
```

The geometry and domain is constructed using the `generate_mesh()` function, via the Mshr interface. It provides a Constructive Solid Geometry interface for geometry, and a mesh resolution parameter as the last argument. The mesh can be then locally refined using the mesh refinement interface in FEniCS.

2.1.1 Questions

a) Set up a model problem where a parabolic inlet with $1m/s$ max flow is given for a rectangular domain of dimension $1m \times 4m$ with a circular obstacle of radius $.1m$ placed with center at the coordinates $x_1 = x_2 = 0.5$.

b) Compare the Midpoint and implicit Euler methods for Navier-Stokes (the PDE defined by the weak residual "r") with the stabilization parameter $d = 0.2h^{3/2}$ what do you see? How is it related to the analytical properties of the methods?

a) Create different meshes and experiment with different refinement levels in order to observe flow characteristics.

2.2 Convection-reaction equations

We will now use the velocity field computed from solving the fluid model to transport some quantity, here the concentrations of two chemicals c_0 and c_1, that is we would now like to solve a system of convection-reaction equations with $N_c = 2$ components (species) where $Z_h = [V_h]^{N_c}$:

$$r_C(c, z) = (\dot{c} + (u \cdot \nabla)c, z) + \kappa((c_0c_1, z_0) + (c_0c_1, z_1)) = 0, \quad \forall z \in Z_h \tag{2}$$

c (Solution: concentration (vector-valued with N_c components)

z (Test function (vector valued with N_c components))

This system models the convection (or transport) of c with the *convective time derivative*: $\dot{c}+(u\cdot\nabla)c$ which arises from an Eulerian choice of coordinates (fixed in space, observing the quantity flowing past with velocity u). The chemical reaction $c_0 + c_1 \rightarrow c_2$, meaning that c_0 reacts with c_1 to form c_2, is modeled by the terms $\kappa((c_0c_1, z_0)+(c_0c_1, z_1))$ with the reaction constant κ determining the reaction rate. We do not represent c_2 in the template model that you are given, so here c_0 and c_1 are simply removed from the system when they react.

In FEniCS notation the model can be written:

```
r_C = (inner(c - c0, z)/k + inner(dot(grad(c), u), z))*dx + kappa*(c[0]*c[1]*z[0] + c[0]*c[1]*z[1])*dx
```

again we add boundary conditions and stabilization:

```
r_C = ((inner(c - c0, z)/k + inner(dot(grad(c), u), z))*dx + kappa*(c[0]*c[1]*z[0] + c[0]*c[1]*z[1])*dx +
    gamma*(im*cm*inner(c[0] - 1., z[0]) + im*(1. - cm)*inner(c[1] - 1., z[1]))*ds + # Weak boundary cond.
    delta*inner(grad(c)*u, grad(z)*u)*dx + delta*inner(grad(c), grad(z))*dx) # Stabilization
```

again in FEniCS we can solve the equation for one timestep simply by:

```
solve(r_C==0, w)
```

and again we have a very similar timestepping loop:

```
t, T = 0., 10. # Time interval
while t < T: # Time-stepping loop

    ... # Solve the Navier-Stokes and convection-reaction PDEs (one timestep)

    t += k; u0 = project(u, V); c0 = project(c, Z) # Shift to next timestep
```

By studying the plots we can get an approximate overview of the reaction process. However, to get a quantitative measure of the reaction, we compute a scalar *functional* of the solution (an integrated quantity). In this case we integrate the sum of the concentrations $M_c = \int_0^T \int_{\partial\Omega_o} \sum_0^{N_c} c_i ds dt$ over the outlet boundary, with the FEniCS notation:

```
Mc = k*om*(c[0] + c[1])*ds; ctot += assemble(Mc)  # Compute total concentration flowing out
```

which is inside the timestepping loop, resulting in an integral over the time interval.

2.2.1 Questions

a) Extend your Navier-Stokes solver to include the convection-reaction PDEs described above.

b) Study how much and how little you can make the chemicals react by modifying the mixing of the fluid. For example, how does the viscosity ν in the fluid model influence the mixing? Can you modify the geometry or other model parameters so that the species fully react (one of them is completely consumed before reaching the outlet)?

c) Add the third species c_2 to the model, by adding reaction terms, and to the plotting, by adding plotting commands to the `plot_compact()` function. The `NC` variable specifies the size of the convection-reaction system (how many species and how many equations that are allocated).

d) Try to invent your own chemical reactions and implement them in the model.

References

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