Lab module 4: The Finite Element Method for Partial Differential Equations - diffusive and convective models

Johan Jansson (jjan@kth.se)

February 28, 2017

0 Jupyter-FEniCS web PDE solver environment

The address of the web Jupyter-FEniCS cloud 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.

You can find example Jupyter notebooks with implementations from the lab under the `src` directory in the zip archive of the lab module, which you can upload in the Jupyter interface.

NB!: The files in the web environment are **not** stored on disk, which means they will disappear if the system is rebooted. 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 jjan/fenics-mooc:test
```

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 [2] framework for automated solution of partial differential equations (PDE) to formulate finite element methods (FEM) that solve PDE and investigate some of the concepts for ODE and PDE in the course.

Specifically you will investigate FEM for fluid flow modeled by the incompressible Navier-Stokes equations [3, 4] and transport and chemical reactions modeled by convection-reaction equations.

The goal of this session is to:

1. Understand how to enforce Dirichlet and Neumann boundary conditions weakly using the Robin formulation.

2. Become familiar and experiment with diffusion, convection and reaction models.

3. Become familiar and experiment with different time-stepping methods here applied to the incompressible 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 [2] there is extensive documentation of the interface at both overview and detail level.

For reference material, please see the lecture notes from the DD1354 Models and Simulation course at KTH [**?**] and the book Computational Differential Equations [1].

2 Exercises

2.1 Convection and Diffusion

As a basic linear model problem we examine the stationary convection-diffusion equation:

$$\begin{aligned} R(u) &= \beta \cdot \nabla u - \nabla \cdot (\epsilon \nabla u) = 0 \\ u &= g_D,\ x \in \Gamma_D \quad \text{Dirichlet boundary condition} \\ -(\nabla u \cdot n &= g_N,\ x \in \Gamma_N \quad \text{Neumann boundary condition} \end{aligned} \tag{1}$$

where $u = u(x)$, $\beta \cdot \nabla u$ *convects* the quantity u by the divergence-free flow velocity β, and $\nabla \cdot (\epsilon \nabla u)$ *diffuses* the quantity with *diffusion coefficient* ϵ.

The weak form of (1) is given by:

$$r(u, v) = (\beta \cdot \nabla u, v) + (\epsilon \nabla u, \nabla v) - ((\nabla u) \cdot n, v)_\Gamma = 0,\ \forall v \in V \tag{2}$$

where we have integrated the diffusion term by parts to generate the boundary term denoted by Γ.

Before we can procced to derive the finite element method for the model, we need to consider how to enfore the boundary conditions. We will learn about a general method for achieving this below.

2.1.1 General Robin boundary conditions

A unified formulation to be able to enforce both the Dirichlet and Neuman is the *Robin* condition:

$$-(\nabla u) \cdot n = \gamma(u - g_D) + g_N,\ x \in \Gamma \tag{3}$$

When $\gamma = 0$ we recover the Neumann condition, and when $\frac{1}{\gamma} = 0$ we recover the Dirichlet condition. The parameter γ can thus be seen as a penalty parameter.

The Robin condition can then be treated just as a Neumann condition, but with the advantage of also being able to model Dirichlet boundary conditions.

To enfore the Robin condition in the weak form, we simply replace the normal derivative boundary term in the weak form $((\nabla u) \cdot n, v)_\Gamma$, by the data in the Robin condition $\gamma(u - g_D) + g_N$. This gives the new weak form:

$$r(u, v) = (\beta \cdot \nabla u, v) + (\epsilon \nabla u, \nabla v) + (\gamma(u - g_D) + g_N, v)_\Gamma = 0,\ \forall v \in V \tag{4}$$

now with enforced Robin boundary conditions.

2.1.2 Finite element method

The finite element method for the model is then simply plugging in a finite element solution $U = \sum_{i=0}^{N} \xi_i \phi_i(x)$ instead of u, and requiring orthogonality against only the finite element basis functions:

$$r(U, v) = (\beta \cdot \nabla U, v) + (\epsilon \nabla U, \nabla v) + (\gamma(U - g_D) + g_N, v)_\Gamma = 0,\ U = \sum_{i=0}^{N} \xi_i \phi_i(x),\ \forall v \in \{\phi_0, ..., \phi_N\} \tag{5}$$

We cannot choose γ arbitrarily large as this will deteriorate the condition number of the matrix. A balanced choice appears to be $\gamma = \frac{10}{h}$ with h the cell size.

2.1.3 FEniCS formulation

A FEniCS formulation of the finite element method for the convection-diffusion model is then simply transcribing into ASCII notation:

```
r = (inner(beta, grad(u))*v + inner(epsilon*grad(u), grad(v)))*dx + inner(gamma*(u - g_D) + g_N, v)*ds
```

To mark different boundary conditions on different parts of the boundary, we may multiply the boundary terms with boundary markers, which are 1 or 0, depending on which part of the subdomain is marked.

2.1.4 Questions

1. You are given a complete FEniCS implementation in `src/Convection-Diffusion.ipynb`. The "Dm1" marker marks the circle boundary inside the box, and the "Dm2" marker marks the left boundary. Enforce a Dirichlet condition on the left boundary with $g_D = 1$, and Dirichlet condition on the circle boundary with $g_D = 0$.

2. Modify the diffusion coefficient, try the values: $1e-2$, $1e-3$ and $1e-4$. What do you observe? We will introduce a method to handle this deficiency of the method in the next section below.

2.2 Timestepping the incompressible Navier-Stokes equations

Now we will study two different timestepping methods: implicit Euler (first order accurate) and the Midpoint method (second-order accurate and energy-conserving, in principle the same method as the Trapezoid method). We will do this for fluid flow, the incompressible Navier-Stokes equations to investigate a realistic case. The timestepping is separated from the spatial discretization, which is more complex and does not have to be fully understood in this lab, but is here fully described to see the generality of the methodology and interface.

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

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

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 penatly 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_C==0, c)
```

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

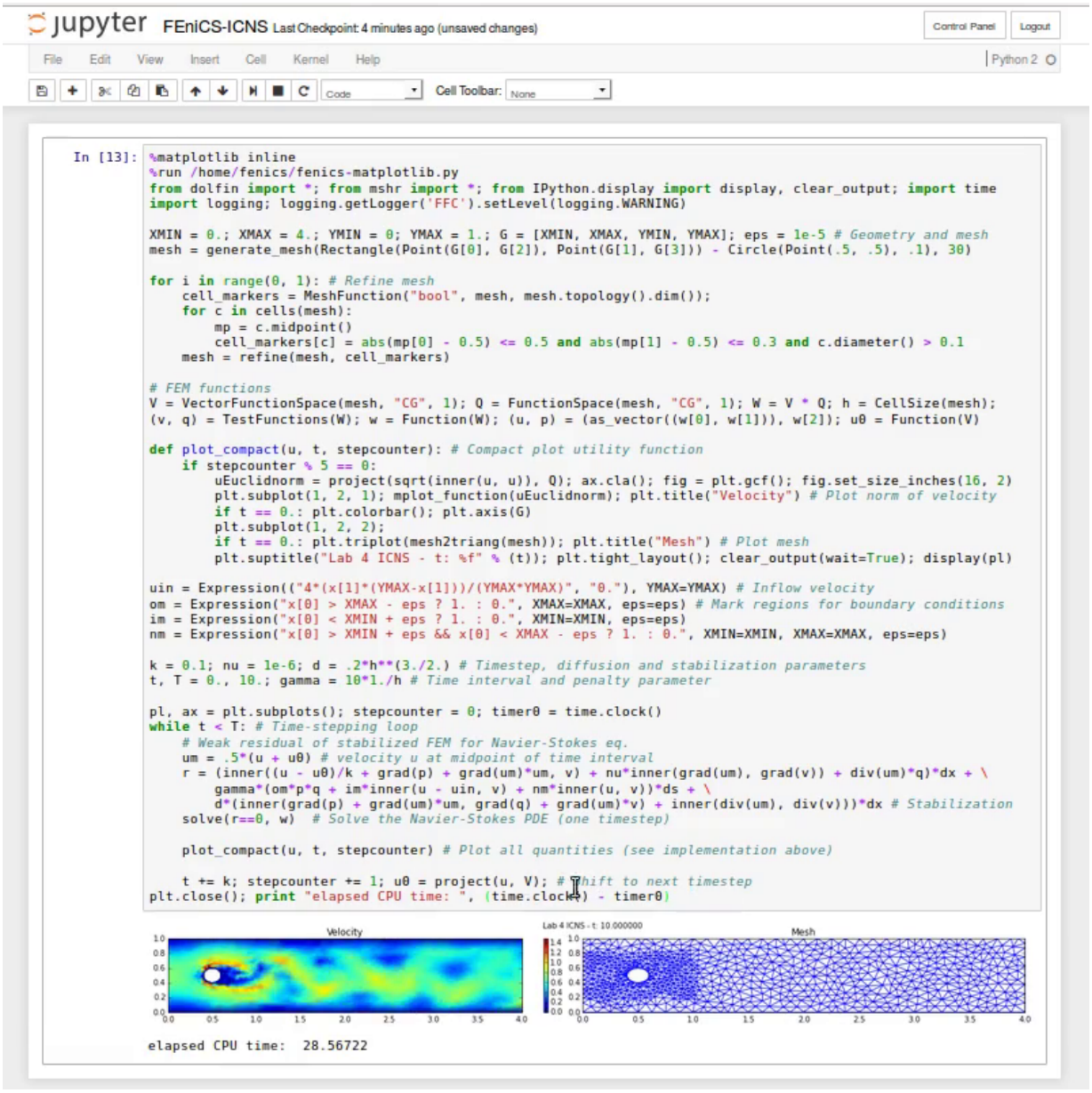
```
t, T = 0., 10. # Time interval
while t < T: # Time-stepping loop

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

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

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

The geometry and domain is constructed using the `generate_mesh()` function, via the Mshr interface. It provides a Constructive Solid Geometry interface for geometry, which we hope is fairly self-explanatory from the template, and a mesh resolution parameter as the last argument. The mesh is then locally refined using the mesh refinement interface in FEniCS.

2.2.1 Screenshot

The template program (here in compact format) and the expected output when running it should look like this:



2.2.2 Questions

a) Implement the Backward Euler timestepping method, and compare the Midpoint and Backward Euler methods for Navier-Stokes (the PDE defined by the weak residual "r") with the parameters given in the template program, what do you see? How is it related to the analytical properties of the methods?

2.3 Convection-reaction equations

We will now use the velocity field computed from solving the fluid model to transport a quantity, here the concentration c of a chemical.

We would now like to solve a system of convection-reaction equations with N_c components (species) where $Z_h = [V_h]^{N_c}$:

$$r_C(c, z) = (\dot{c} + (u \cdot \nabla)c, z) + \gamma((c_0 c_1, z_0) + (c_0 c_1, z_1)) = 0, \quad \forall z \in Z_h \tag{7}$$

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 $\gamma((c_0c_1, z_0) + (c_0c_1, z_1))$ with the reaction constant γ 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 + gamma*(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 + gamma*(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==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; # Shift to next timestep
   u0 = project(u, V);
   c0 = project(c, Z)
```

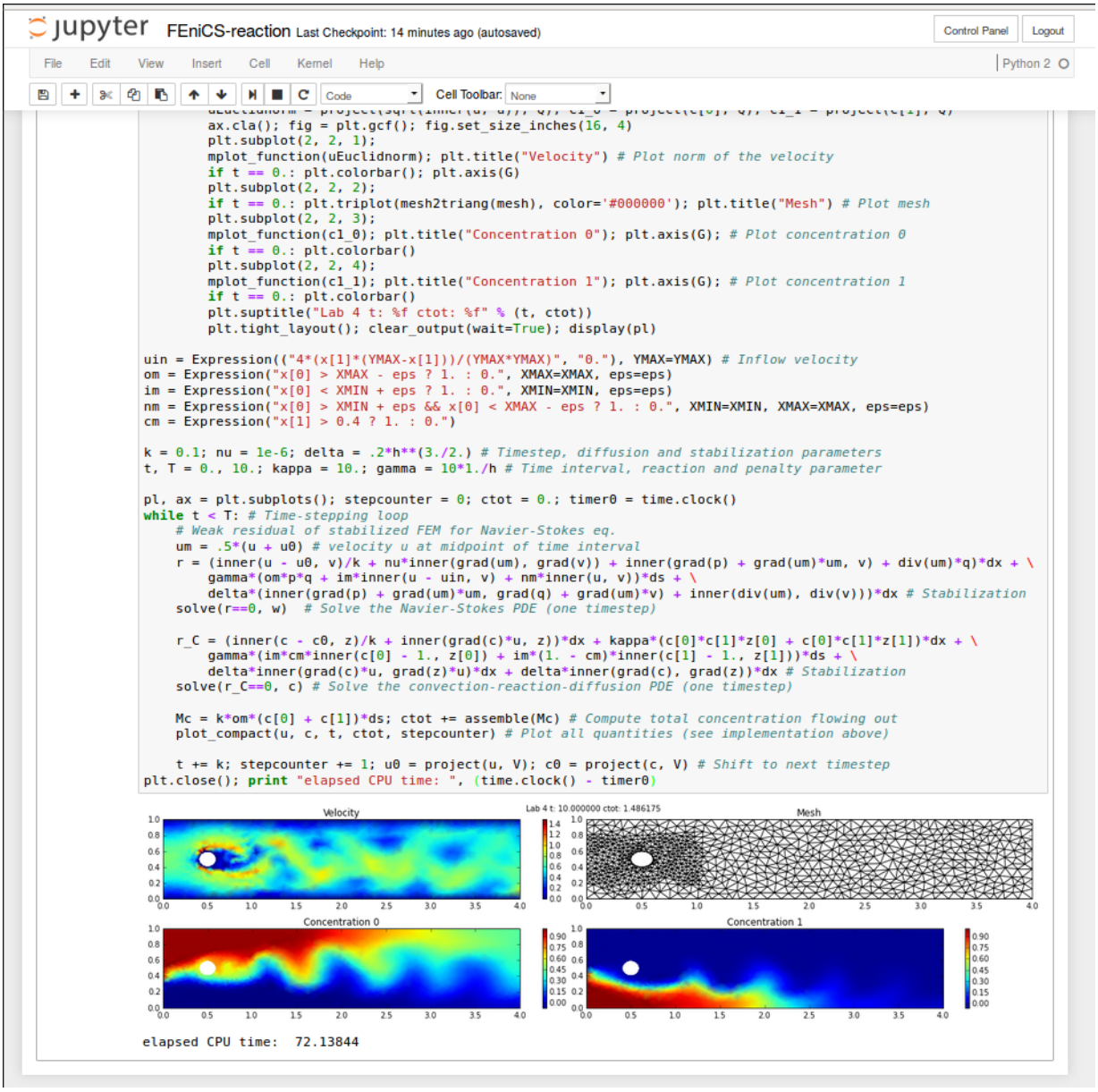
By studying the plots we can get an approximate overview of the reaction process. However, to get a quantitative measure of the reaction, we compute a scalar *functional* of the solution (an integrated quantity). In this case we integrate the sum of the concentrations $M_c = \int_0^T \int_{\partial\Omega_o}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

2.3.1 Screenshot

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



2.3.2 Questions

a) 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)?

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

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

References

[1] Kenneth Eriksson, Don Estep, Peter Hansbo, and Claes Johnson. *Computational Differential Equations*. Cambridge University Press New York, 1996.

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

[3] 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.

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