# INF5620 - Mandatory Project 2.

Peter Killingstad

November 4, 2016

## Project 2: Nonlinear diffusion equation

Investigate various numerical aspects of the nonlinear diffusion model:

$$\varrho u_t = \nabla \cdot (\alpha(u)\nabla u) \ + \ f(\boldsymbol{x}, t)$$

with initial condition $u(\boldsymbol{x}, 0) = I(x)$ and boundary condition $\partial u/\partial n = 0$. The coefficient $\varrho$ is constant and $\alpha$ is a known function of u.

**a)** Introduce some finite difference approximation in time that leads to an implicit scheme (say a Backward Euler, Crank-Nicolson, or 2-step backward scheme). Derive a variational formulation of the initial condition and the spatial problem to be solved at each time step (and at the first time step in case of a 2-step backward scheme).

With Backward Euler method the differential equation becomes:

$$u_t \approx \frac{u^n - u^{n-1}}{\Delta t} = \frac{1}{\varrho}(\nabla \cdot (\alpha(u^n)\nabla u^n) \ + \ f(\boldsymbol{x}, t_n))$$

$$u^{n-1} = u^n - \frac{\Delta t}{\varrho}(\nabla \cdot (\alpha(u^n)\nabla u^n) \ + \ f(\boldsymbol{x}, t_n))$$

Denote $u$ for $u^n$ and $u^{(1)}$ for $u^{n-1}$. The variational form becomes: find $u \in V$ such that

$$\int_\Omega (u - \frac{\Delta t}{\varrho}\nabla \cdot \alpha(u)\nabla u - \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t_n) - u^{(1)})v \, d\Omega = 0$$

The spatial finite element problem involves second order derivatives. Apply integration by parts on the term

$$\int_\Omega (\frac{\Delta t}{\varrho}\nabla \cdot (\alpha(u)\nabla u))v \, d\Omega$$

so that with homogeneous Neumann boundary conditions the the variational form becomes:

$$\int_\Omega (uv + \frac{\Delta t}{\varrho}\alpha(u)\nabla u \cdot \nabla v - \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t_n)v - u^{(1)}v) \, d\Omega = 0.$$

with $u = \Sigma_j c_j \psi_j$ and $v = \psi_i$.

**b)** Formulate a Picard iteration method at the PDE level, using the most recently computed $u$ function in the $\alpha(u)$ coefficient. Derive general formulas for the entries in the linear system to be solved in each Picard iteration. Use the solution at the previous time step at initial guess for the Picard iteration.

Write the variational form as:

$$F_i = \int_\Omega (uv + \frac{\Delta t}{\varrho}\alpha(u)\nabla u \cdot \nabla v - \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t)v - u^{(1)}v) \, d\Omega = 0.$$

Picard iteration needs a linearization where $u^-$ is the most recent approximation to $u$ in $\alpha$ such that

$$F_i \approx \hat{F}_i = \int_\Omega (uv + \frac{\Delta t}{\varrho}\alpha(u^-)\nabla u \cdot \nabla v - \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t)v - u^{(1)}v)\, d\Omega = 0.$$

The equations $\hat{F} = 0$ are now linear and a system on the form

$$\Sigma_{j \in I_s} A_{i,j} c_j = b_i, i \in I_s$$

to solve for the unknown $c_j$ can be derived by inserting $u = \Sigma_j c_j \psi_j$ and $v = \psi_i$ such that

$$A_{i,j} = \int_\Omega (uv + \frac{\Delta t}{\varrho}\alpha(u^-)\nabla u \cdot \nabla v)\, d\Omega, \quad b_i = \int_\Omega \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t_n)v + u^{(1)}v)\, d\Omega = 0.$$

As stated above, $u^-$ is the most recent approximation to $u$. So for this problem, start by setting $u^- = u^{(1)} = u^k$ for an iteration k. Then seek a new improved solution $u^{k+1}$ in iteration $k + 1$ such that $u^{k+1} \to u$ as $k \to \infty$.

**c)** Restrict the Picard iteration to a single iteration. That is, simply use a $u$ value from the previous time step in the $\alpha(u)$ coefficient. Implement this method with the aid of the FEniCS software (in a dimension-independent way such that the code runs in 1D, 2D, and 3D).

With a single iteration $\hat{F}$ becomes:

$$\int_\Omega (uv + \frac{\Delta t}{\varrho}\alpha(u^{(1)})\nabla u \cdot \nabla v - \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t)v - u^{(1)}v)\, d\Omega = 0.$$

The implementation of the variatonal form in the Python program

```
u_k = u_1
F = u*v*dx + inner(dt/rho*alpha(u_k)*nabla_grad(u),nabla_grad(v))*dx
 - u_1*v*dx - dt/rho*f*v*dx
a = lhs(F)
L = rhs(F)
```

To express that there really is a difference between $u^-$ and $u^{(1)}$, these values is explicitly denoted as:

```
u_1.assign(u_)
u_k.assign(u_)
```

when it iterates over time.

**d)** The first verification of the FEniCS implementation may reproduce a constant solution. Find values of the input data , $\alpha$, $f$, and $I$ such that $u(x, t) = C$, where $C$ is some choosen constant.

Values of the input data is choosen as :

$$\begin{aligned} \varrho &= 1.0 \\ \alpha &= 1 \\ f &= 0 \\ I &= 5 \end{aligned}$$

which in fact gives $u(x,t) = C = 5$.

**e)** The second verification of the FEniCS implementation may reproduce a simple analytical solution of the PDE problem. Assume $\alpha(u) = 1$, $f = 0$, $\Omega = [0,1] \times [0,1]$, P1 elements and $I(x,y) = \cos(\pi x)$.
A model for an error measure:

$$E = K_t \Delta t + K_x \Delta x^2 + K_y \Delta y^2 = Kh$$

if $h = \Delta t = \Delta x^2 = \Delta y^2$ is a common discretization measure. A suitable measure E can be taken as the discrete L2 norm of the solution at the nodes, computed by:

```
e = (u_e.vector().array() - u_.vector().array())
E = numpy.sqrt(numpy.sum(e**2)/u_.vector().array().size)
```

For some fixed point in time, $E/h$ remains approximately constant as the mesh in space and time is simultaneously refined:

$$\begin{aligned} h = 0.01000, \quad E/h = 1.7884550e-03 \\ h = 0.00444, \quad E/h = 1.6318174e-03 \\ h = 0.00250, \quad E/h = 1.5413329e-03 \\ h = 0.00111, \quad E/h = 1.5179406e-03 \\ h = 0.00040, \quad E/h = 1.5024606e-03 \end{aligned}$$

**f)** The analytical solution in the test above is valid only for a linear version of the PDE without a source term $f$. To get an indication whether the implementation of the nonlinear diffusion PDE is correct or not, use the method of manufactured solutions. Restrict the problem to one space dimension, $\Omega = [0,1]$, and choose

$$u(x,t) = t \int_0^x q(1-q)\, dq = tx^2(\frac{1}{2} - \frac{x}{3})$$

and $\alpha(u) = 1 + u^2$. The given $f(x,t)$ is

$$f(x,t) = -\varrho\frac{x^3}{3} + \varrho\frac{x^2}{2} + 8t^3\frac{x^7}{9} - 28t^3\frac{x^6}{9} + 7t^3\frac{x^5}{2} - 5t^3\frac{x^4}{4} + 2tx - t.$$

Comparing the FEniCS solution and the given $u$ by computing a suitable L2 norm:

```
e = (u_e.vector().array() - u_.vector().array())
E = numpy.sqrt(numpy.sum(e**2)/u_.vector().array().size)
```

yields a result of E/h which remains approximately constant as the mesh in space and time is simultaneously refined:

$$
\begin{aligned}
h = 0.01000, \quad E/h = 1.5793614e-03 \\
h = 0.00250, \quad E/h = 1.5589363e-03 \\
h = 0.00111, \quad E/h = 1.5459262e-03 \\
h = 0.00040, \quad E/h = 1.5392876e-03 \\
h = 0.00010, \quad E/h = 1.5364506e-03
\end{aligned}
$$

**g)** There is different sources of numerical errors in the FEniCS program. First of all, the discretization of the differential equation will contribute to errors. As the differential equation is non linear, picard iteration is used to linearize the problem. For a not very small $\Delta t$, a single Picard iteration, that is used in this case, contributes with an error that will pollute the error model assumed in convergence tests. In the general case with a not sufficiently small $\Delta t$, one must perform Picard iterations until the iterations reach a tolerance significantly smaller than the discretization errors. However, as $\Delta t \to 0$, the error associated with a single Picard iteration may get significantly less. Numerical integration in FEniCS will also lead to numerical errors that can theoretically pollute the solution. However, as those errors is of the same order or smaller than the discretization, the impact is not important.