Project 2: Nonlinear diffusion equation

Greger Svenn

November 30, 2015

Given the diffusion model

$$\varrho u_t = \nabla \cdot (\alpha(u)\nabla u) + f(\mathbf{x}, t) \tag{1}$$

with initial condition $u(\mathbf{x}, 0) = I(x)$ and boundary condition $\partial u/\partial n = 0$. Where ϱ is a constant and $\alpha(u)$ is a known function.

a)

Using backward euler in time we get

$$\begin{aligned} \varrho\frac{u^{n+1} - u^n}{\Delta t} &= \nabla \cdot (\alpha(u^{n+1})\nabla u^{n+1}) + f(\mathbf{x}, t_{n+1}) \\ u^{n+1} - \frac{\Delta t}{\varrho}(\nabla \cdot (\alpha(u^{n+1})\nabla u^{n+1}) + f(\mathbf{x}, t_{n+1})) &= u^n \end{aligned} \tag{2}$$

To get the variational form we multiply with v and take the inner product

$$\int_0^L u^{n+1}v\ dx - \frac{\Delta t}{\varrho}\left(\int_0^L \nabla\cdot(\alpha(u^{n+1})\nabla u^{n+1})v\ dx + \int_0^L f(\mathbf{x}, t_{n+1})v\ dx\right) = \int_0^L u^n v\ dx \tag{3}$$

By using integration by parts on the diffusion term, we get a more suitable function

$$\int_0^L \nabla \cdot (\alpha(u^{n+1})\nabla u^{n+1})v\ dx = [v\nabla u^{n+1}]_0^L - \int_0^L \alpha(u^{n+1})\nabla u^{n+1} \cdot \nabla v\ dx \tag{4}$$

where the term $[v\nabla u^{n+1}]_0^L$ is zero because of the boundary condition, $\partial u/\partial n = 0$. The variational formulation of the diffusion model is now

$$\int_0^L u^{n+1}v\ dx + \frac{\Delta t}{\varrho}\left(\int_0^L \alpha^{n+1}\nabla u^{n+1} \cdot \nabla v\ dx - \int_0^L f^{n+1}v\ dx\right) = \int_0^L u^n v\ dx \tag{5}$$

We are also asked to derive a variational formulation of the initial condition

$$\begin{aligned} u(\mathbf{x}, 0) &= u^{n=0} = I(x) \\ \int_0^L u^0 v\ dx &= \int_0^L I(x)v\ dx \end{aligned} \tag{6}$$

b)

In (5) we've got the non-linear term $\alpha(u^{n+1})\nabla u^{n+1} \cdot \nabla v$. To linearize this term we use the picard iteration, saying that u^- is an approximation of u. By inserting u^- into function α we get a linear term, since we imply that u^- is known making $a(u^-) = constant$.

$$\int_0^L u^{n+1}v\ dx + \frac{\Delta t}{\varrho}\left(\int_0^L \alpha(u^-)\nabla u^{n+1} \cdot \nabla v\ dx - \int_0^L f^{n+1}v\ dx\right) = \int_0^L u^n v\ dx \tag{7}$$

A good approximation for u^- is using the previous, known, time step. $u^- = u^n$.

$$\int_0^L u^{n+1} v \, dx + \frac{\Delta t}{\varrho}\Big(\int_0^L \alpha^n \nabla u^{n+1} \cdot \nabla v \, dx - \int_0^L f^{n+1} v \, dx \Big) = \int_0^L u^n v \, dx \tag{8}$$

c)

We need to implement (8)in FEniCS, where we have used Picard iteration to approximate $\alpha(u^{n+1})$ as $\alpha(u^n)$.
Rearranging of (8) gives us

$$\int_0^L u^{n+1} v \, dx + \frac{\Delta t}{\varrho} \int_0^L \alpha^n \nabla u^{n+1} \cdot \nabla v \, dx = \int_0^L (u^n v \, dx + \frac{\Delta t}{\varrho} f^{n+1}) v \, dx \tag{9}$$

From my solver file, `solver.py`, the implementation is done by

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

where `a` is the left hand side of (9), and `L` is the right hand side.

d)

As a first verification of our FEniCS-solver we want to reproduce a constant solution. The values of ϱ, α, f and I need to be adapted such that

$$u(\mathbf{x}, t) = C. \tag{10}$$

From our original diffusion equation, (1), u_t and ∇u equals zero. This means that our source term, f, also needs to be zero. ϱ and α may be an arbitrary real number, and $I(x) = u(\mathbf{x}, 0) = C$.

The resulting errors is

```
--------P1 element--------
----------1 D----------
Max error, t=0.50: 0.000
Max error, t=1.00: 0.000
----------2 D----------
Max error, t=0.50: 0.000
Max error, t=1.00: 0.000
----------3 D----------
Max error, t=0.50: 0.000
Max error, t=1.00: 0.000
--------P2 element--------
----------1 D----------
Max error, t=0.50: 0.000
```

```
Max error , t=1.00: 0.000
----------2 D----------
Max error , t=0.50: 0.000
Max error , t=1.00: 0.000
----------3 D----------
Max error , t=0.50: 0.000
Max error , t=1.00: 0.000
```

The error is always zero. Verification OK!

e)

As a second verification of my FEniCS implementation, we want to reproduce

$$u(x, y, t) = e^{-\pi^2 t} cos(\pi x). \tag{11}$$

Because I've used backward euler discretization the error in time is $\mathcal{O}(\Delta t)$. Furthermore we use $h = \Delta t = \Delta x^2 = \Delta y^2$ as a discretization measure.
As we simultaneously refine our mesh in both time and space we want to compare E/h $(= K)$ at the same time point, where

$$E = (K_t + K_x + K_y)h = Kh \tag{12}$$

```
h=0.70711, E/h=0.13567, t=0.50
h=0.50000, E/h=0.00587, t=0.50
h=0.35355, E/h=0.00568, t=0.50
h=0.25000, E/h=0.00557, t=0.50
h=0.17678, E/h=0.00544, t=0.50
h=0.12500, E/h=0.00536, t=0.50
```

From the result above we can see that E/h is approximately the same for smaller h values.

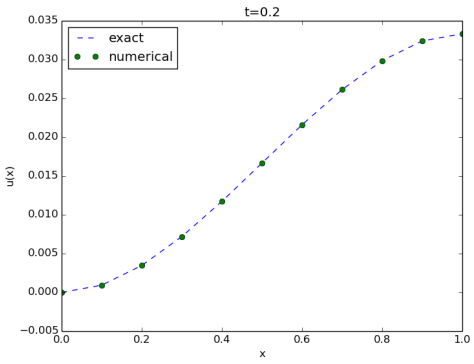
f)

To get an indication whether the implementation of the nonlinear diffusion PDE is correct or not, we can use the method of manufactured solutions. Say we restrict the problem to one space dimension, and choose

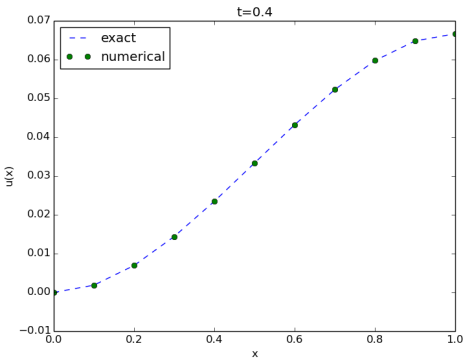
$$u(x, t) = \int_0^x q(1 - q)dq = tx^2\left(\frac{1}{2} - \frac{x}{3}\right) \tag{13}$$

and $\alpha(u) = 1 + u^2$. The source term is given, calculated using `sympy`.

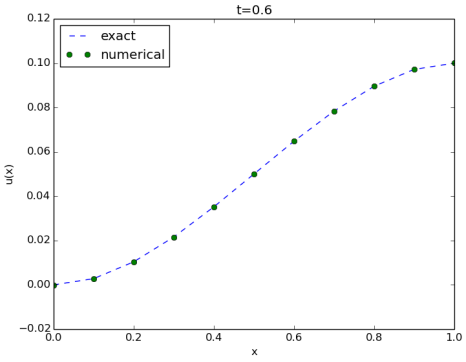
(a) Nonlinear diffusion t=0.2



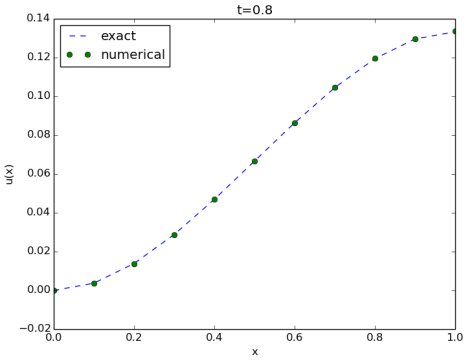
(b) Nonlinear diffusion t=0.4



(c) Nonlinear diffusion t=0.6



(d) Nonlinear diffusion t=0.8



Implementation of the nonlinear diffusion PDE is correct

g)

For a not very small Δt, the single Picard iteration contributes with an error that will pollute the error model assumed in convergence tests. However, as $\Delta t \to 0$, the error associated with a single Picard iteration may get significantly less than the (also small) discretization errors such that correct convergence rate can be obtained.
Numerical integration in FEniCS will also lead to errors that can theoretically pollute the measured convergence rates.

h)

To verify (13) by checking the convergence rate, we must eliminate the error due to a single Picard iteration. By finding a manufactured solution, using α from the previous time step, that solves the PDE will eliminate this error. The source term is calculated using `sympy`.

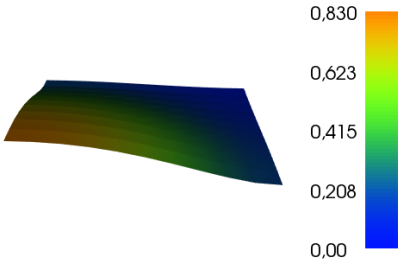
The convergence rate is calculated by

$$r = \frac{log(|E_{max}^{n+1} - E_{max}^{n}|)}{log(|h^{n+1} - h^{n}|)} \tag{14}$$

```
h=0.040000,    convergence rate:3.93070640022
h=0.002400,    convergence rate:0.95778243801
h=0.000300,    convergence rate:0.902968969894
h=0.000020,    convergence rate:1.03299682122
h=0.000002,    convergence rate:0.958060395549
```

i)

(a) Gaussian function at t=0



(b) Full diffusion, at t=0.565