NonlinearDiffusionEquation

Milad H. Mobarhan

December 2, 2013

1 Project 3: Nonlinear diffusion equation

Summary. The goal of this project is to discuss various numerical aspects of a nonlinear diffusion model.

1.1 Mathematical problem

We look at the PDE problem

$$\begin{aligned} \varrho u_t &= \nabla\cdot(\alpha(u)\nabla u) + f(\boldsymbol{x},t), && t>0 \\ u(\boldsymbol{x},0) &= I(\boldsymbol{x}), && \boldsymbol{x}\in\Omega \\ \alpha(u(\boldsymbol{x},t))\frac{\partial}{\partial n}u(\boldsymbol{x},t) &= 0, && \boldsymbol{x}\in\partial\Omega_N \end{aligned}$$

The coefficiet ϱ is constant and $\alpha(u)$ is a known function of u.

1.2 Discretization in time

First we need a mesh in time, here taken as uniform with mesh points $t_n = n\Delta t$, $n = 0,1,\ldots,N_t$. A Backward Euler scheme consists of sampling at t_n and approximating the time derivative by a backward difference

$$[D_t^- u]^n \approx (u^n - u^{n-1})/\Delta t.$$

This approximation turns our PDE into a differential equation that is discrete in time, but still continuous in space. With a finite difference operator notation we can write the time-discrete problem as

$$[D_t^- \varrho u = \nabla\cdot(\alpha(u)\nabla u) + f(\boldsymbol{x},t)]^n\,.$$

which gives the nonlinear time-discrete PDEs

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

or with $u^n = u$ and $u^{n-1} = u_1$:

$$u - \frac{\Delta t}{\varrho}\nabla\cdot(\alpha(u^n)\nabla u) - \frac{\Delta t}{\varrho}f(\boldsymbol{x}, t_n) = u_1\,.$$

From the last equation we can define the residual:

$$R = u - \frac{\Delta t}{\varrho}\nabla\cdot(\alpha(u)\nabla u) - \frac{\Delta t}{\varrho}f(\boldsymbol{x}, t_n) - u_1\,.$$

1.3 The variational form

The least-squares principle is equivalent to demanding the error to be orthogonal to the space V when approximating a function f by $u \in V$. With a differential equation we do not know the true error so we must instead require the residual R to be orthogonal to V. This idea implies seeking $\{c_i\}_{i\in\mathcal{I}_s}$ such that

$$(R, v) = 0, \quad \forall v \in V\,.$$

This statement is equivalent to R being orthogonal to the $N + 1$ basis functions only:

$$(R, \psi_i) = 0, \quad i \in \mathcal{I}_s,$$

resulting in $N + 1$ equations for determining $\{c_i\}_{i\in\mathcal{I}_s}$. The variational form for our specific case is given by

$$\begin{aligned}
\int_\Omega (u\psi_i - \frac{\Delta t}{\varrho}\nabla\cdot(\alpha(u^n)\nabla u)\psi_i - \frac{\Delta t}{\varrho}f(\boldsymbol{x}, t_n)\psi_i - u_1\psi_i)\,\mathrm{d}x = 0\\
\int_\Omega (u\psi_i - \frac{\Delta t}{\varrho}\left[-\int_\Omega \alpha(u)\nabla u\cdot\nabla\psi_i\,\mathrm{d}x + \int_{\partial\Omega}\alpha(u)\frac{\partial u}{\partial n}\psi_i\,\mathrm{d}x\right] - \frac{\Delta t}{\varrho}f(\boldsymbol{x}, t_n)\psi_i - u_1\psi_i)\,\mathrm{d}x = 0\\
\int_\Omega (u\psi_i + \frac{\Delta t}{\varrho}\alpha(u)\nabla u\cdot\nabla\psi_i - \frac{\Delta t}{\varrho}f(\boldsymbol{x}, t_n)\psi_i - u_1\psi_i)\,\mathrm{d}x = 0
\end{aligned}$$

or more compactly

$$(u, \psi_i) + \frac{\Delta t}{\varrho}(\alpha\nabla u, \nabla\psi_i) = (u_1\psi_i) + \frac{\Delta t}{\varrho}(f^n, \psi_i)$$

Initial condition

The variational form for the initial condition is found by expanding

$$u(\boldsymbol{x}, 0) = \sum_j c_j^0\psi_j(\boldsymbol{x}) = I(\boldsymbol{x})$$

The Galerkin method implies

$$\left(\sum_j c_j^0\psi_j(\boldsymbol{x}) - I(\boldsymbol{x}), \psi_i\right) = 0, \quad i \in \mathcal{I}_s,$$

or

$$\sum_j (\psi_j, \psi_i) c_j^0 = (I, \psi_i)$$

1.4 Picard iteration method at the PDE level

Our aim is to discretize the problem in time and then present techniques for linearizing the time-discrete PDE problem "at the PDE level" such that we transform the nonlinear stationary PDE problems at each time level into a sequence of linear PDE problems, which can be solved using any method for linear PDEs. In our case we have

$$u = \frac{\Delta t}{\varrho} \nabla \cdot (\alpha(u) \nabla u) + \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t_n) + u_1$$

which is nonlinear beacuse of the dependency on u in the variable coefficient α. Picard iteration needs a linearization where we use the most recent approximation u_- to u in α:

$$u = -\frac{\Delta t}{\varrho} \nabla \cdot (\alpha(u_-) \nabla u) + \frac{\Delta t}{\varrho} f(\boldsymbol{x}, t_n) + u_1$$

In variational form is given by

$$(u, \psi_i) + \frac{\Delta t}{\varrho} (\alpha(u_-) \nabla u, \nabla \psi_i) = (u_1 \psi_i) + \frac{\Delta t}{\varrho} (f^n, \psi_i)$$

1.5 Convergence test

The first verification of the FEniCS implementation may assmue $\alpha(u) = 1$, $f = 0$, $\Omega = [0,1] \times [0,1]$, P1 elements, and $I(x,y) = \cos(\pi x)$. The exact solution is then $u(x,y,t) = e^{-\pi^2 t} \cos(\pi x)$. The error in space is then $\mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$, while the error in time is $\mathcal{O}(\Delta t^p)$, with $p = 1$ for the Backward Euler scheme and $p = 2$ for the Crank-Nicolson or the 2-step backward schemes. We set $h = \Delta t^p = \Delta x^2$ and perform a convergence rate study as h is decreased, using Backward Euler scheme:

```
h =  0.1       E =  0.000497030720246  r =  1.22455181028
h =  0.05      E =  0.000212694571666  r =  1.66702631777
h =  0.025     E =  6.69778929328e-05  r =  1.60581512224
h =  0.0125    E =  2.20055871886e-05  r =  1.09776873673
h =  0.00625   E =  1.02818589919e-05  r =  1.03661750021
h =  0.003125  E =  5.01208793551e-06  r =  1.0386373625
```

where $r = E/h$, which is approximately constant, as expected.

1.6 Manufactored solution

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, $\Omega = [0,1]$, and choose

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

and $\alpha(u) = 1 + u^2$. The following sympy session computes an $f(x,t)$ such that the above u is a solution of the PDE problem:

In [36]:
```
from sympy import *

def a(u):
    return 1 + u**2

def u_simple(x, t):
    return x**2*(Rational(1,2) - x/3)*t

x, t, p, dt = symbols('x[0] t p dt')
for x_point in 0, 1:
     print 'u_x(%s,t):' % x_point,
     print diff(u_simple(x, t), x).subs(x, x_point).simplify()

print 'Initial condition:', u_simple(x, 0)

u = u_simple(x, t)
f = p*diff(u, t) - diff(a(u)*diff(u, x), x)
print ccode(f.simplify())
```

```
u_x(0,t): 0
u_x(1,t): 0
Initial condition: 0
-p*pow(x[0], 3)/3 + p*pow(x[0], 2)/2 + 8*pow(t, 3)*pow(x[0], 7)/9 -
28*pow(t, 3)*pow(x[0], 6)/9 + 7*pow(t, 3)*pow(x[0], 5)/2 - 5*pow(t,
3)*pow(x[0], 4)/4 + 2*t*x[0] - t
```

We get the following results for different final times:

```
T =  0.1  E =  3.7822919294e-08
T =  0.5  E =  1.86763505521e-06
T =  1.0  E =  9.02389336352e-06
T =  2.0  E =  4.29791921834e-05
T =  3.0  E =  0.000131418978921
```

For much higher values of T, the error become more significant, which is due to the error in the single Picard iteration.

1.7 Sources of numerical errors

- Truncation error from approximating the function with piecewise linear elements.
- Error due to a single Picard iteration
- Time discretisation (not explicitly due to the FEniCS)