The second obligatory project

IN5270

2018

Project: Nonlinear diffusion equation

Project size: approximately 35 h. This particular project will have significant overlap with topics for the final exam.

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

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

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

Required background knowledge for this project.

1. Discretization of PDEs by finite differences in time and finite element in space.
2. Solution of linear diffusion equations with FEniCS.
3. Theory of methods for solving nonlinear PDEs, in particular how to define Picard iterations on a variational form.
4. Verification procedures: testing against exact solutions and measuring convergence rates.

a) Use Backward Euler discretization (finite differencing) in the time direction to construct an implicit scheme. Derive a variational formulation of the initial condition and the spatial problem to be solved at each time step.

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 as initial guess for the Picard iteration.

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).

d) The first verification of the FEniCS implementation may reproduce a constant solution. Find values of the input data ϱ, α, f, and I such that $u(\boldsymbol{x}, t) = C$, where C is some chosen constant. Write test functions that verify the computation of a constant solution in 1D, 2D, and 3D for P1 elements (use simple domains: interval, square, box).

e) The second verification of the FEniCS implementation may reproduce a simple analytical solution of the PDE problem. 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. We can then write a model for an error measure:

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

if $h = \Delta t^p = \Delta x^2 = \Delta y^2$ is a common discretization measure and $K = (K_t + K_x + K_y)$. A suitable measure E can be taken as the discrete L_2 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 of time, say $t = 0.05$. In this code segment, `u_e` is a projection of the exact solution onto the function space used for `u`.

Show that E/h remains approximately constant as the mesh in space and time is simultaneously refined (i.e., h is reduced).

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, we can use the *method of manufactured solutions*. Say we restrict the problem to one space dimension, $\Omega = [0,1]$, and choose

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

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:

```
>>> from sympy import *
>>> x, t, rho, dt = symbols('x t rho dt')
>>>
>>> def a(u):
...     return 1 + u**2
...
>>> def u_simple(x, t):
```

```
...     return x**2*(Rational(1,2) - x/3)*t
...
>>> # Show that u_simple satisfies the BCs
>>> 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()
...
u_x(0,t): 0
u_x(1,t): 0
>>> print 'Initial condition:', u_simple(x, 0)
Initial condition: 0
>>>
>>> # MMS: full nonlinear problem
>>> u = u_simple(x, t)
>>> f = rho*diff(u, t) - diff(a(u)*diff(u, x), x)
>>> print f.simplify()
-rho*x**3/3 + rho*x**2/2 + 8*t**3*x**7/9 - 28*t**3*x**6/9 +
7*t**3*x**5/2 - 5*t**3*x**4/4 + 2*t*x - t
```

Compare the FEniCS solution and the u given above as a function of x for a couple of t.

Remark.

A convergence rate test with the manufactured solution above meets fundamental problems, because in convergence tests we assume that the temporal and spatial discretization errors are the only errors. 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.

In the general case with a not sufficiently small Δt, we must perform Picard iterations and use a tolerance for stopping the iterations that is significantly smaller than the discretization errors.

(Numerical integration in FEniCS will also lead to errors that can theoretically pollute the measured convergence rates, but the error in numerical integration formulas is of the same order or smaller than the discretization errors so impact of numerical integration is not important.)

g) List the different sources of numerical errors in the FEniCS program.

h) (Optional.) To verify the nonlinear PDE implementation in FEniCS by checking convergence rate, we must eliminate the error due to a single Picard iteration. This can be done by finding a manufactured solution that fulfills the PDE with $\alpha(u^{(1)})$, where $u^{(1)}$ is the solution at the previous time step. Choosing the manufactured solution (1) and $\alpha(u) = 1 + u^2$, the following `sympy` session computes the necessary source term f:

```
>>> u_1 = u_simple(x, t-dt)
>>> f = rho*diff(u, t) - diff(a(u_1)*diff(u, x), x)
```

```
>>> print simplify(f)
rho*x**2*(-2*x + 3)/6 -
(-12*t*x + 3*t*(-2*x + 3))*(x**4*(-dt + t)**2*(-2*x + 3)**2 + 36)/324
- (-6*t*x**2 + 6*t*x*(-2*x + 3))*(36*x**4*(-dt + t)**2*(2*x - 3)
+ 36*x**3*(-dt + t)**2*(-2*x + 3)**2)/5832
```

Perform a convergence rate test by decreasing $h = \Delta t^p = \Delta x^2$. (Observe that the error E is proportional to h, or assume $E \sim h^r$ and compute r from two values of h and E and observe that $r \to 1$.)

i) (Optional.) Simulate the nonlinear diffusion of Gaussian function. Due to symmetry of the Gaussian function (with respect to $x = 0$ and $y = 0$), it suffices to simulate one quarter of the domain:

$$I(x, y) = \exp\left(-\frac{1}{2\sigma^2}\left(x^2 + y^2\right)\right), \quad (x, y) \in \Omega = [0, 1] \times [0, 1],$$

where σ measures the width of the Gaussian function. Choose $\alpha(u) = 1 + \beta u^2$, where β is a constant one can play around with. The boundary conditions in this project, $\partial u/\partial n = 0$ are compatible with symmetry conditions on $x = 0$ and $y = 0$, while at $x = 1$ and $y = 1$ we assume a wall so the diffused substance cannot escape from the domain (which means $\partial u/\partial n = 0$).