Project 2

Non-linear diffusion equation

Håkon V. Treider, Christian Fleischer

November, 2015

1 Introduction

In this project, we are going to solve the following non-linear diffusion equation (in 1, 2 and 3 dimensions).

$$\varrho \frac{\partial u}{\partial t} = \nabla \cdot (\alpha(u) \nabla u) + f(\vec{x}, t) \tag{1}$$

We have the initial distribution, (condition) $u(\vec{x}, t = 0) = I(\vec{x})$ and boundary condition $\frac{\partial u}{\partial n} = 0$, where n is a vector normal to the edge. The coefficient ϱ is a constant, while $\alpha(u)$ is a known function of u. The term ∇u is the gradient of u, while $\nabla \cdot g(u)$ is the divergence of the function $g(u) = \alpha \nabla u$.

(a) Finding the variational form

For this project, we chose to use Backward Euler, the standard implicit scheme, for the discretization of equation 1, with respect to time. This gave us:

$$\varrho \frac{\partial u}{\partial t} \approx \varrho \Big(\frac{u^n - u^{n-1}}{\Delta t} \Big) = \nabla \cdot \left(\alpha \left(u^n \right) \nabla u^n \right) + f^n \tag{2}$$

Next up, we need to derive a variational formulation of the initial condition and the spatial problem to be solved at each time step. We start by defining

$$R = u^n - u^{n-1} - \frac{\Delta t}{\varrho} \Big[\nabla \cdot \left(\alpha \left(u^n \right) \nabla u^n \right) + f^n \Big] \tag{3}$$

Using the Galerkin approach with function space $\mathbf{V}$, we multiply both sides with v and integrate over the domain Ω, which, for instance, in the 1D-case, typically is something like $\Omega = [0, L]$

$$\int_\Omega u^n v \,\mathrm{d}\vec{x} = \int_\Omega \left(u^{n-1} + \frac{\Delta t}{\varrho}\Big[\nabla \cdot (\alpha\,(u^n)\,\nabla u^n) + f^n\Big]\right) v \,\mathrm{d}\vec{x}, \quad \forall v \in \mathbf{V} \tag{4}$$

We can't have second order derivatives, so we need to use integration by parts to take care of this. We also make use of the divergence theorem, and rewrite $\alpha(u^n) \to \alpha^n$, to make things simpler:

$$\int_\Omega \nabla \cdot (\alpha^n \nabla u^n) v \,\mathrm{d}\vec{x} = -\int_\Omega \Big(\alpha^n \nabla u^n \cdot \nabla v\Big) \,\mathrm{d}\vec{x} + \cancel{\int_{\partial\Omega} \Big(\alpha^n \nabla u^n \cdot \vec{n} v\Big) \,\mathrm{d}\vec{x}} \tag{5}$$

$$= -\int_\Omega \Big(\alpha^n \nabla u^n \cdot \nabla v\Big) \,\mathrm{d}\vec{x} \tag{6}$$

...where $\partial\Omega$ is the boundary of the domain, and $\vec{n}$ is a vector normal to the domain at the boundary, pointing outwards (away). The dot product of the gradient of u and $\vec{n}$, is by definition, (Neumann boundary conditions), zero at the boundary. Thus the whole term vanishes. We may now write down the whole equation on variational form, where we have renamed $u^n \to u$ and $u^{n-1} \to u^{(1)}$:

$$\int_\Omega uv \,\mathrm{d}\vec{x} = \int_\Omega \left(u^{(1)} v - \frac{\Delta t}{\varrho}\Big(\alpha \nabla u \cdot \nabla v - fv\Big)\right) \mathrm{d}\vec{x} \tag{7}$$

When dealing with the very first time step, we may simply replace $u^{(1)}$ with $I(\vec{x})$ in the above equation.

(b) Picard iteration

The term

$$\frac{\Delta t}{\varrho} \alpha \nabla u \cdot \nabla v \tag{8}$$

from equation 7 is non-linear for u since α is dependent on u. In order to solve equation 7 we want to first make it linear. We do this by replacing u in α with u^-, where u^- is an approximation of the unknown u. We then get the equation

$$\int_\Omega uv \,\mathrm{d}\vec{x} = \int_\Omega \left(u^- v - \frac{\Delta t}{\varrho}\Big(\alpha(u^-) \nabla u \cdot \nabla v - fv\Big)\right) \mathrm{d}\vec{x} \tag{9}$$

For the first iteration we set u^- to be value of the unknown at the previous time-step, i.e $u^- = u^{(1)}$. We then solve equation 9, set $u^- = u$ for the next iteration, and repeat this for each iteration. For a given iteration (after the first) u^- is then set to be the solution of the previous iteration. We stop iterating when some stopping criteria is fulfilled. This is typically either when

$$|u - u^-| \leq \epsilon_u,$$

where ϵ_u is a chosen tolerance, or when the residual of the equation is smaller than a chosen tolerance ϵ_r.

(c) Solving the variational problem in FEniCS

For a given time step $t = n$, the equation we are trying to solve is now a linear equation, since we approximate the dependant variable u in α from $\alpha(u^{n+1}) \to \alpha(u^n)$, or with the new naming convention $\alpha(u) \to \alpha(u^{(1)})$. With a slight rearranging of terms, we end up with the following solvable equation:

$$\int_\Omega \left(uv + \frac{\Delta t}{\varrho} \Big(\alpha(u^{(1)}) \nabla u \cdot \nabla v \Big) \right) \, \mathrm{d}\vec{x} = \int_\Omega \Big(u^{(1)} - f \Big) \, v \, \mathrm{d}\vec{x} \tag{10}$$

$$\int_\Omega a(u, v) \, \mathrm{d}\vec{x} = \int_\Omega L(v) \, \mathrm{d}\vec{x} \tag{11}$$

In the program code, this is translated to:

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

(d) Constant solution

A test that is easy to implement, is to see whether a constant solution is reproduced correctly after the solver is done with it. If we insert $u(\vec{x}, t) = C$ into equation 1, we get $f(\vec{x}, t) = 0$. This means we can choose an arbitrary value for ϱ, since it disappears anyway. The same goes for the function $\alpha(u)$; we may choose any function we'd like. Since nothing will be changed, the only thing that matters for the ending result, is the initial condition, we must set this to the desired value $I(\vec{x}) = C$

The size of the resulting errors are of the order 10^{-12} to 10^{-15} which indicate a correct implementation of the solver.

(e) Testing a simple analytical solution

A little more advanced test is an analytical solution where we test if linear equations work properly. We use $\alpha = 1$, and set the source term to zero, $f = 0$. The initial condition is set to $I(x, y) = \cos(\pi x)$ which gives an analytical solution:

$$u_{\text{exact}}(x, y, t) = e^{-\pi^2 t} \cos(\pi x) \tag{12}$$

Depending on the choice of scheme, the error should scale as

$$E = K_t \Delta t^p + K_x \Delta x^2 + K_y \Delta y^2 = Kh \tag{13}$$

under the condition that $h = \Delta t^p = \Delta x^2 = \Delta y^2$ where $K = K_t + K_x + K_y$. The value of p depends on the finite difference scheme used to discretize the

derivatives in time. As stated earlier, we used backward Euler so in our case, $p = 1$.

In the program code, we test this by decreasing $h = \Delta t$, while enlarging the mesh in accordance with the formula for h above. The different test runs was run until ending time $T = 0.1$ was reached. The results can be seen in table 1. We see that E/h stays almost constant, as was expected.

Table 1: Error scaling, when testing analytical solution

h	E/h	N
0.1000	0.1488	3
0.0500	0.1267	4
0.0250	0.2034	6
0.0125	0.1817	9
0.0063	0.1109	13
0.0031	0.1076	18
0.0016	0.1582	25

(f) Testing a manufactured solution

The previous analytical solution is valid only for a linear version of the diffusion equation we started with (remember $\alpha = 1$, and $f = 0$). To get an indication whether the implementation of the nonlinear diffusion PDE is correct or not, we can use the method of manufactured solutions. We will only do this in one spatial dimension. We use:

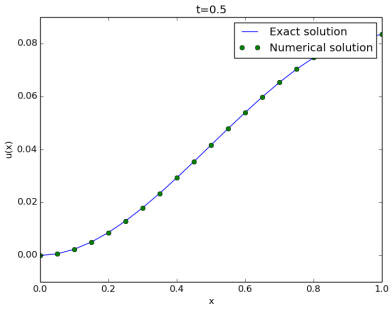
$$u(x,t) = tx^2\left(\frac{1}{2} - \frac{x}{3}\right) \tag{14}$$

$$\alpha(u) = 1 + u^2 \tag{15}$$

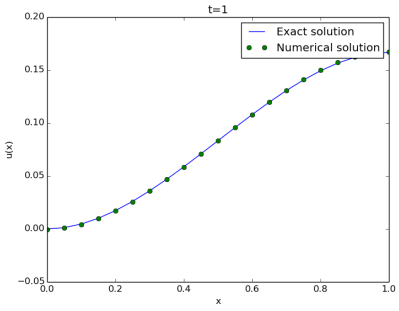
We then ask `SymPy` to create an appropriate source term $f(x,t)$. We get the following equation that we implement in our program:

$$f(x,t) = -\varrho\frac{x^3}{3} + \varrho\frac{x^2}{2} + 8t^3\,\frac{x^7}{9} - 28t^3\,\frac{x^6}{9} + 7t^3\,\frac{x^5}{2} - 5t^3\,\frac{x^4}{4} + 2tx - t \tag{16}$$

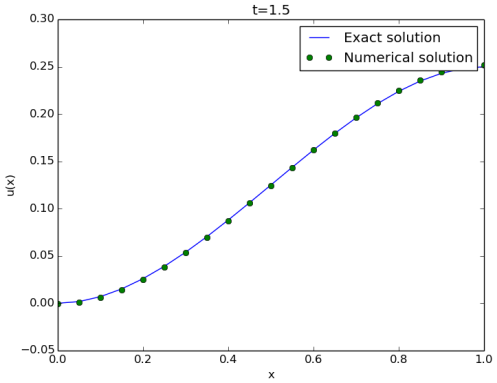
We run and compare the analytic solution with our numerical solution from FEniCS and plot the results. This can be seen in figure 1



(a) After T=0.5



(b) After T=1.0



(c) After T=1.5

Figure 1: Comparison of analytic solution with non-linear PDE.

(g) Numerical errors

1. Discretization errors in space and time; $\mathcal{O}(\Delta x^2)$, $\mathcal{O}(\Delta y^2)$, $\mathcal{O}(\Delta z^2)$, $\mathcal{O}(\Delta t)$.
2. We get an error from numerical integration in FEniCS. This error is of the same order as discretization errors or smaller.
3. The single Picard iteration introduces an error, which is dependent on Δt.

(h) Convergence rates

We have the exact solution

$$u(x,t) = tx^2\left(\frac{1}{2} - \frac{x}{3}\right). \tag{17}$$

From equation 17 we see that our initial condition has to be $I(x) = u(x, 0) = 0$. We use P1 elements and $\varrho = 1$, with the α and f given in the assignment. Starting with $\Delta t = 0.5$ and halving Δt for each run, we get the convergence rates listed in table 2. We see that $r \to 1$, which is expected since Backward Euler has a discretization error in time $\mathcal{O}(h)$.

Table 2: The table lists the convergence rates r, for the step lengths $h1$ and $h2$.

$h1$	$h2$	r
0.500000	0.250000	1.118764
0.250000	0.125000	1.017141
0.125000	0.062500	1.012774
0.062500	0.031250	1.011533
0.031250	0.015625	1.009206
0.015625	0.007812	1.007698
0.007812	0.003906	1.005399
0.003906	0.001953	1.003954
0.001953	0.000977	1.003418

(i) Diffusion of Gaussian function

Using $\sigma = 0.5$ we get the initial state shown in figure 2. We use P1 elements, $f = 0$, and the parameters $\varrho = 1$ and $\beta = 10$. We set $\Delta t = 0.002$ with end time $T = 0.2$, and reach the equilibrium state shown in figure 3.

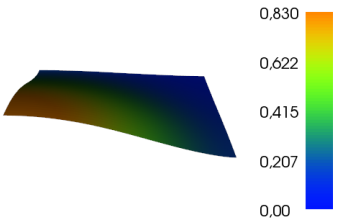


Figure 2: This figure shows the initial state of the Gaussian function for one quarter of the domain.

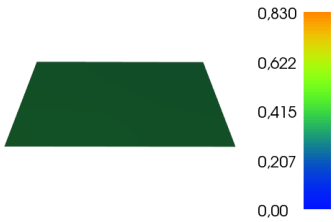


Figure 3: This figure shows the equilibrium state of the diffused Gaussian function.