INF5620

Arnfinn Mihle Paulsrud

December 17, 2012

1 exercise a

We have been given the non-linear PDE

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

with initial condition $u(\mathbf{x}, 0) = I(\mathbf{x})$ and boundary condition, Neumann conditions, $\partial u/\partial n = 0$. ρ is a constant and $\alpha(u)$ is a known function of u.
We'll first derive a variational formulation of the initial system, and we'll use Backward Euler scheme in time, on the u_t part.

$$u_t = \frac{u^n - u^{n-1}}{\Delta t}$$

this is the Backward Euler scheme. By inserting this into eq.(1), we get

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

To get the variational form, we have to take the integral on both sides of the equation over some region, Ω, and multiply each sides by v.

$$\int_\Omega \rho u^n v - \Delta t\nabla\cdot(\alpha(u^n)\nabla u^n)v d\mathbf{x} = \int_\Omega \rho u^{n-1}v + \Delta t f(\mathbf{x}, t_n)v d\mathbf{x} \tag{2}$$

If we now use integration by parts on $-\Delta t\nabla\cdot(\alpha(u^n)\nabla u^n)v$, and use our boundary condition $\partial u/\partial n = 0$, we get

$$-\Delta t\int_\Omega \underbrace{\nabla\cdot(\alpha(u^n)\nabla u^n)}_{u'} v d\mathbf{x} = \underbrace{\int_{d\Omega}\frac{\partial u}{\partial n}v ds}_{=0} + \Delta t\int_\Omega \alpha(u^n)\nabla u^n\cdot\nabla v d\mathbf{x}$$

If we insert this into eq.(2), we get

$$\int_\Omega \rho u^n v + \Delta t\alpha(u^n)\nabla u^n\cdot\nabla v d\mathbf{x} = \int_\Omega \rho u^{n-1}v + \Delta t f(\mathbf{x}, t_n)v d\mathbf{x}$$

we now rename $u^n \rightarrow u$, $u^{n-1} \rightarrow u_1$ and $f(\mathbf{x}, t_n) \rightarrow f^n$ and use inner product notation

$$\rho(u, v) + \Delta t(\alpha(u)\nabla u, \nabla v) = \rho(u_1, v) + \Delta t(f^n, v)$$

We see that our left side depends on the unknowns u and v, and our left side depends only on the unknown v. We can now rewrite our function to two functions $a(u, v)$ and $L(v)$

$$a(u, v) = (\rho(u, v) + \Delta t(\alpha(u)\nabla u, \nabla v))dx \tag{3}$$

and

$$L(v) = (\rho(u_1, v) + \Delta t(f^n, v))dx \tag{4}$$

For the first time step we have to use our initial condition $u(\mathbf{x}, 0) = I(\mathbf{x})$, if we insert this into eq.(4), we get

$$L_0(v) = (\rho(I(\mathbf{x}), v) + \Delta t(f^n, v))dx \tag{5}$$

$a(u, v)$ is the same for all time steps.

2 exercise b

Picard iteration is used for making non-linear PDE's behave like linear PDE's. In our case this is done by using u from the last time step in $\alpha(u)$. This means that we now have a known expression for $\alpha(u)$. With Picard iteration eq.(3) becomes

$$a(u, v) = (\rho(u, v) + \Delta t(\alpha(u_1)\nabla u, \nabla v))dx \tag{6}$$

and eq.(4) is unchanged

$$L(v) = (\rho(I(\mathbf{x}), v) + \Delta t(f^n, v))dx \tag{7}$$

The Picard iterations

```
u = Function(V)
eps = 1.0
tol = 1.0E-5
iter = 0
maxiter = 25

while eps > tol and iter < maxiter:
    iter += 1
    solve(a == L, u, bcs)
    diff = u.vector().array() - u_k.vector().array()
    eps = numpy.linalg.norm(diff, ord=numpy.Inf)
    u_k.assign(u)
```

3 exercise c

If we just drop the while-loop from b, we get a Picard iteration scheme with one iteration.

```
u = Function(V)
solve(a == L, u, bcs)
u_k.assign(u)
```

So we just assign u_k to be u. This will probably yield a high error. The way we can solve the problem easy for both 1D, 2D and 3D without much problem is to implement this bit code

```
degree = int(sys.argv[1])
divisions = [int(arg) for arg in sys.argv[2:]]
d = len(divisions)
domain_type = [UnitInterval, UnitSquare, UnitCube]
mesh = domain_type[d-1](*divisions)
```

4 exercise d

To get the information at $t = 0.05$ we use this if test

```
if t > (0.05 - dt) and t < (0.05 + dt):
    e = ue.vector().array() - u.vector().array()
    E = numpy.sqrt(numpy.sum(e**2)/u.vector().array().size)
    print 'error = ', E, 't = ', t, 'E/h = ', E/dt
```

And the results for mesh size 20, 40, 60, 80 and 100 are

error = 0.00221418885556t = 0.05E/h = 0.885675542224
error = 0.000552450360912t = 0.05E/h = 0.883920577459
error = 0.000244994642726t = 0.05E/h = 0.881980713815

error = 0.0001376178238t = 0.05E/h = 0.880754072322
error = $8.79935868532e - 05$t = 0.05E/h = 0.879935868532

respectively, and as we can see E/h is quite stable.

5 exercise e

In this exercise we where going to compare our code with a manufactured solution

$$u(x,t) = t\int_0^x q(1-q)dq = tx^2\left(\frac{1}{2} - \frac{x}{3}\right) \tag{8}$$

The way we do this is to find some $f(x,t)$ that satisfy the manufactured solution. This $f(x,t)$ is given as

$$-\rho x^3 + \frac{\rho x^2}{2} + \frac{8t^3x^7}{9} - \frac{28t^3x^6}{9} + \frac{7t^3x^5}{2} - \frac{5t^3x^4}{4} + 2tx - t \tag{9}$$

In our python program we just use the Expression function

```
f = Expression('-rho*pow(x[0],3)/3.0 + rho*pow(x[0],2)/2.0\
                                + pow(t,3)*(8.0*pow(x[0],7)/9.0 - 28.0*pow(x[0],6)/9.0 +
                                     7.0*pow(x[0],5)/2.0 - 5.0*pow(x[0],4)/4.0)\
                                + t*(2.0*x[0] - 1.0)', rho = rho, t=0.0)
```

The results from the simulation is given in figure 1 and 2. As we can see the shape of the line is the same for the manufactured solution and the numerical, but the starting point and end point is not exactly the same. The reason for this might be because of numerical precision.

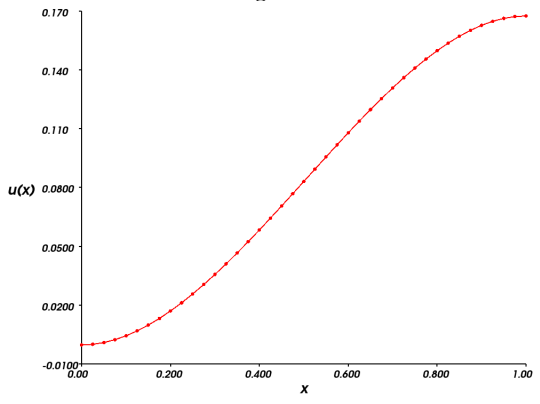

Figure 1: The plot shows the numerical solution of eq. (8)

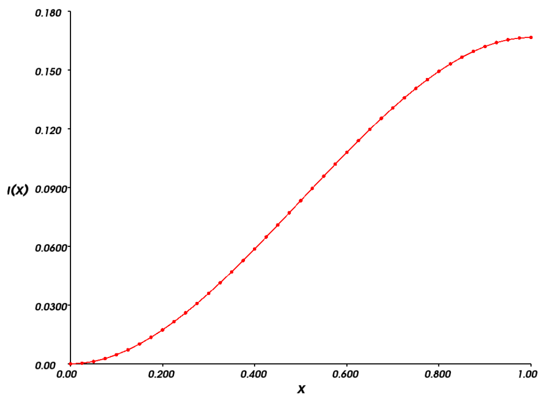

Figure 2: The plot shows the exact solution of eq. (8)

6 exercise f

When we are doing numerics on a computer there will always be some numerical errors due to numerical precision.
We have also used Backward Euler which have a truncation error in time $\mathcal{O}(\Delta t)$.

7 exercise h

In this exercise we where going to simulate the nonlinear diffusion of Gaussian function

$$I(x, y) = e^{-\frac{1}{2\sigma^2}\left(x^2+y^2\right)} \tag{10}$$

As we can see from figure 3, 4 and 5, the Gaussian is diffusing out into the plane and eventually all the grid points will have the same value of $u(x, y)$.

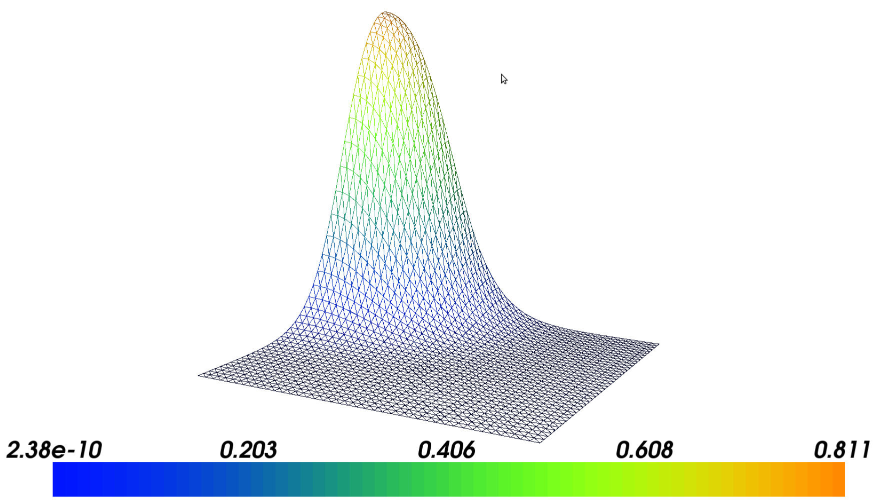

Figure 3: The plot shows the the two dimensional Gaussian function (10) at the star of the simulation for $\sigma = 50$

8 exercise i

In this exercise we are going to apply the group finite element method to the $\alpha(u)$ coefficient. From earlier we have found the expression

$$\int_\Omega \rho u^n v + \Delta t \alpha(u^n) \nabla u^n \cdot \nabla v - \rho u^{n-1} v + \Delta t f^n v dx \tag{11}$$

We now apply the group finite element method to the $\alpha(u)$

$$\alpha(u) = \alpha\left(\sum_j u_j \varphi_i()x\right) \approx \sum_{j=1}^{n} \alpha(u_j)\varphi_j \tag{12}$$

We also write u as linear combinations of the basis V

$$u = \sum_k U_k \varphi_k \tag{13}$$

we can now insert these two equations back into our integral

$$\int_\Omega \rho \sum_k c_k \varphi_k^n v + \Delta t \sum_{j=1}^{n} \alpha(u_j^n)\varphi_j \sum_k c_k \nabla \varphi_k^n \cdot \nabla v - \rho \sum_k c_k \varphi_k^{n-1} v + \Delta t f^n v dx \tag{14}$$

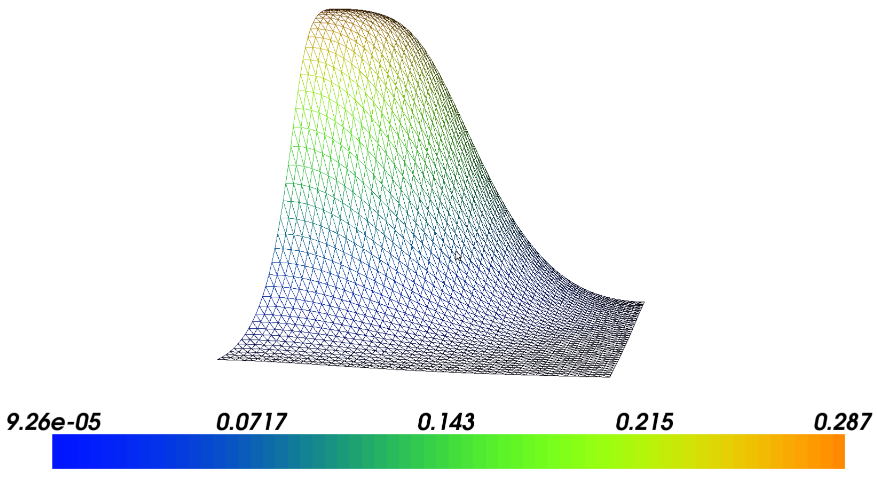

Figure 4: The plot shows the the two dimensional Gaussian function (10) after $50dt$ for $\sigma = 50$

9 exercise j

An other way to solve the nonlinear discretized equation is to use Newtons method.

$$F_i \equiv \int_\Omega \rho(u^n - u^{n-1})v + \Delta t\alpha(u^n)\nabla u^n \cdot \nabla v - \Delta t f^n v dx \tag{15}$$

we now use $u = \sum_j u_j\varphi_j$ and $v = \varphi_i$

$$F_i \equiv \int_\Omega \rho(\sum_j u_j^n\varphi_j - u^{n-1})\varphi_i + \Delta t\alpha(\sum_l u_l^n\varphi_l)\sum_j \nabla u_j^n\varphi_j \cdot \nabla\varphi_i - \Delta t f^n\varphi_i dx \tag{16}$$

$$F_i \equiv \sum_j \int_\Omega \rho(u_j^n\varphi_j - u^{n-1})\varphi_i dx + \Delta t\sum_j \int_\Omega \alpha(\sum_l u_l^n\varphi_l)\nabla u_j^n\varphi_j \cdot \nabla\varphi_i dx - \Delta t\int_\Omega f^n\varphi_i dx \tag{17}$$

We then use $\sum_j \frac{\partial F_i}{\partial u_j}\delta u_j = -F_i$

$$\frac{\partial F_i}{\partial u_j} \equiv \int_\Omega \rho\varphi_j\varphi_i dx + \Delta t\sum_j \int_\Omega \alpha'(\sum_l u_l^n\varphi_l)\nabla\sum_j(u_j^n\varphi_j) \cdot \nabla\varphi_i + \alpha(\sum_l u_l^n\varphi_l)\nabla\varphi_j \cdot \nabla\varphi_i dx \tag{18}$$

This is how far I got...

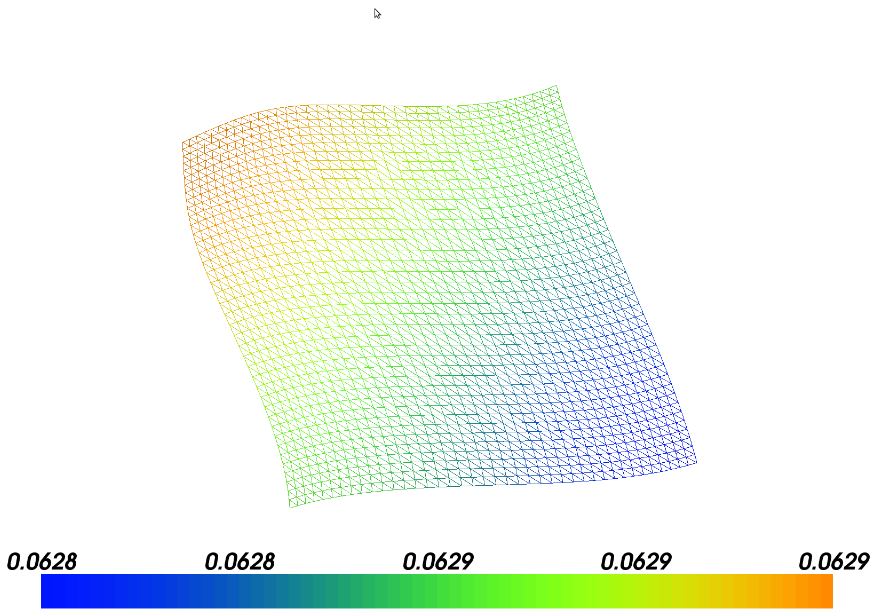

Figure 5: The plot shows the the two dimensional Gaussian function (10) at the end of the simulation after $50dt$ for $\sigma = 50$