Final project INF5620

Mathilde Nygaard Kamperud

December 13, 2012

Introduction

This final project in INF5620 is supposed to be a self manufactured assignment that sums up the curriculum of the course. There has been given a "default"-project about a nonlinear diffusion model:

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

with initial condition $u(x,0) = I(x)$, and boundary condition $\frac{\partial u}{\partial n} = 0$
I have chosen to follow the default-project.
This assignment will be implemented with the aid of the FEniCS software. The text written in itallic font is from the default final project-description.

a)

Introduce some finite difference approximaton in time that leads to an implicit scheme. Derive a variational formulation of the initial condition and the spatial problem to be solved at each time step.
I choose the Backward Euler-scheme. Equation (1) at timestep t^n.

$$\frac{\partial u}{\partial t}|_{t_n} = \left[\nabla \cdot (\alpha(u^n)\nabla u^n + f(x,t^n)\right]/\rho$$

The Backward Euler sceme:

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

To get this equation in a variational form, we multiply with a test function v and integrate over the spatial domain.

$$\int_\Omega u^n v dx^3 - \frac{\Delta t}{\rho}\int_\Omega \left(\nabla \cdot (\alpha(u^n)\nabla u^n)\right) v dx^3 = \int_\Omega u^{n-1} v dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n) v dx^3 \tag{2}$$

We calculate the second integral on the left hand side by integration by parts

$$I_2 = \int_\Omega \left[\nabla \cdot (\alpha(u^n)\nabla u^n)\right] v dx^3 = \int_{d\Omega} \alpha(u^n)\frac{\partial u}{\partial n} v ds - \int_\Omega \alpha(u^n)\nabla u^n \cdot \nabla v dx^3$$

The first term on the right hand side is zero because of the boundary condition $\frac{\partial u}{\partial n} = 0$.

$$I_2 = -\int_\Omega \alpha(u^n)\nabla u^n \cdot \nabla v dx^3$$

Insert this into eq. (2):

$$\int_\Omega u^n v dx^3 + \frac{\Delta t}{\rho}\int_\Omega \alpha(u^n)\nabla u^n \cdot \nabla v dx^3 = \int_\Omega u^{n-1} v dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n) v dx^3$$

We can write this on a weak form

$$a(u^n, v) = L(v)$$

where

$$a(u^n, v) = \int_\Omega u^n v dx^3 + \frac{\Delta t}{\rho}\int_\Omega \alpha(u^n)\nabla u^n \cdot \nabla v dx^3$$
$$L(v) = \int_\Omega u^{n-1} v dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n) v dx^3$$

The function $L(v)$ contains u^{n-1}, but that is known because it is the solution for the previous timestep.
To find u at the first time-step we set u^0 equal to the initial condition I(x). That gives us

$$L_0(v) = \int_\Omega I(x) v dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^0) v dx^3$$

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.
For each time-step we will do Picard iteration, also known as the method of successive substitutions.
We use a known previous solution in $\alpha(u_k^n)$, and solve the equation for u_{k+1}^n

$$u_{k+1}^n - \frac{\Delta t}{\rho}\nabla\cdot(\alpha(u_k^n)\nabla u_{k+1}^n) = u^{n-1} + \frac{\Delta t}{\rho} f(x,t^n) \tag{3}$$

At a timestep n, we do Picard iterations where we solve for u_{k+1} and set the u_k equal to the newly found u_{k+1}, and repeat. We start with $u_0^n = u^{n-1}$. Note that the right hand side does not change as we increase k. We hope that u_{k+1}^n is close to u^n after just a few iterations. Iteration with final k equal to 2:

$$u_1^n - \frac{\Delta t}{\rho}\nabla\cdot(\alpha(u^{n-1})\nabla u_1^n) = u^{n-1} + \frac{\Delta t}{\rho} f(x,t^n) \text{Solve and find } u_1 \tag{4}$$
$$u_2^n - \frac{\Delta t}{\rho}\nabla\cdot(\alpha(u_1^n)\nabla u_2^n) = u^{n-1} + \frac{\Delta t}{\rho} f(x,t^n) \text{Insert } u_1^n \text{ in } \alpha\text{, then solve and find } u_2$$
$$u_3^n - \frac{\Delta t}{\rho}\nabla\cdot(\alpha(u_2^n)\nabla u_3^n) = u^{n-1} + \frac{\Delta t}{\rho} f(x,t^n) \text{Insert } u_2^n \text{ in } \alpha\text{, then solve and find } u_3$$

Now we hope that u_3^n is close to the exact solution u^n, and we move on to the next timestep where we repeat the Picard iteration. In the next timestep we use u_3^n as our u^{n-1}.
We solve these equations by writing them on variational form, and solve by the aid of FEniCS. The variational form becomes

$$\begin{aligned} a(u,v) &= \int_\Omega u_{k+1}^n v dx^3 + \int_\Omega \alpha(u_k^n)\nabla u_{k+1}^n \cdot \nabla v dx^3 \\ L(v) &= \int_\Omega u^{n-1}v + \frac{\Delta t}{\rho} f(x,t^n) v dx^3 + \int_{d\Omega} \alpha(u_k^n)\frac{\partial u_{k+1}^n}{\partial n} v ds \\ &= \int_\Omega u^{n-1}v + \frac{\Delta t}{\rho} f(x,t^n) v dx^3 \end{aligned}$$

c)

Restrict the Picard iteration to a single iteration.
This means that we should stop the iteration after solving the first equation (eq. (5)), and move on to the next timestep.
The weak form:

$$\begin{aligned} a(u,v) &= \int_\Omega u^n v dx^3 + \int_\Omega \alpha(u^{n-1})\nabla u^n \cdot \nabla v dx^3 \\ L(v) &= \int_\Omega u^{n-1}v + \frac{\Delta t}{\rho} f(x,t^n) v dx^3 \end{aligned}$$

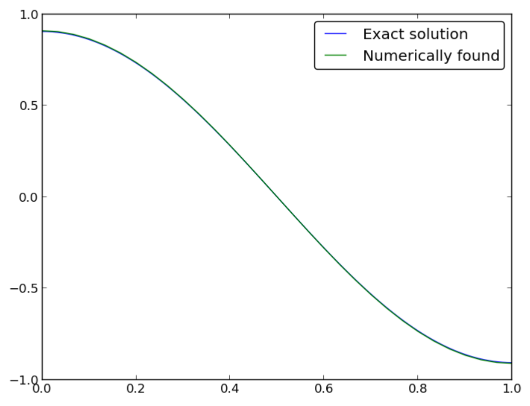
d)

The first verification of the FEniCS implementation may assume $\alpha(u) = 1$ and $f = 0.\Omega = [0,1] \times [0,1]$ 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 $O(\Delta x^2) + O(\Delta y^2)$, while the error in time is $O(\Delta t^p)$, with p=1 for the Backward Euler- scheme. Perform a convergence rate study as $h = \Delta t^p = \Delta x^2$ decreases.
The implemented code for picard iteration with a single iteration can be found in nonlinear_diffusion_solver.py. The method convergence solves a problem (with initial condition given by the user) for a number of decreasing h values. For every timestep the method finds the error, computed as given in the exercise text. The total error for each h-value is found by averaging over the errors found for each time step. The method prints out a list where each listelement is a tuple of size two. The first element of this tuple is the fraction E_{tot}/h, and the second element is just h. Convergence can be used for any dimension. In this test run I ran the program for ten h-values between 0.25 and 0.00028. The result can be found in table 1.

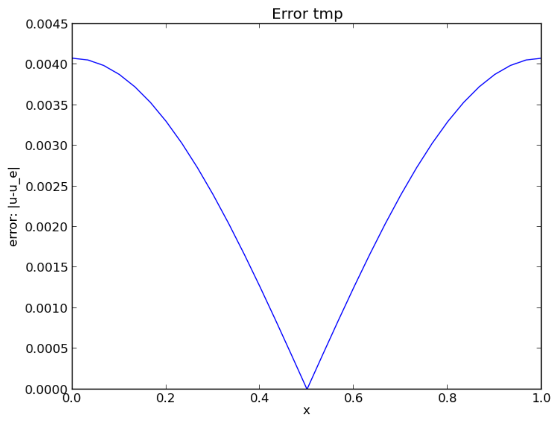
Table 1: Table showing the results of the convergence rate study.

E/h	h
0.0655	0.2500
0.0874	0.1111
0.0995	0.0400
0.1010	0.0100
0.1007	0.0059
0.1002	0.0035
0.0996	0.0011
0.0996	0.0006
0.0990	0.00028

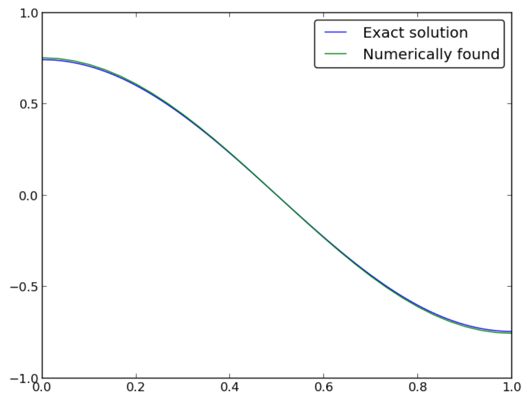
I also plottet the solution and the error for $h = 0.01$. These plots is in figure 1.



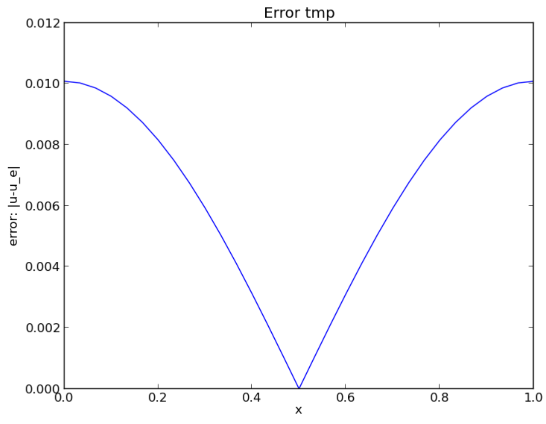
(a) After 1 timestep



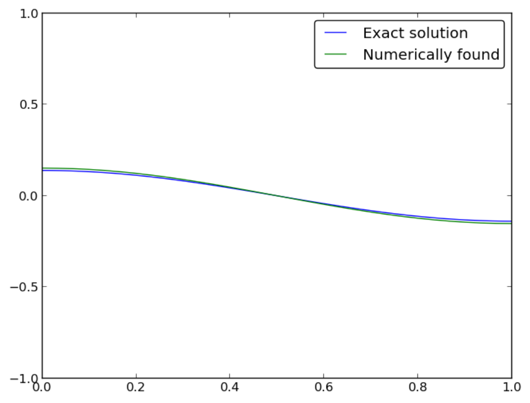
(b) Error after 1 timestep



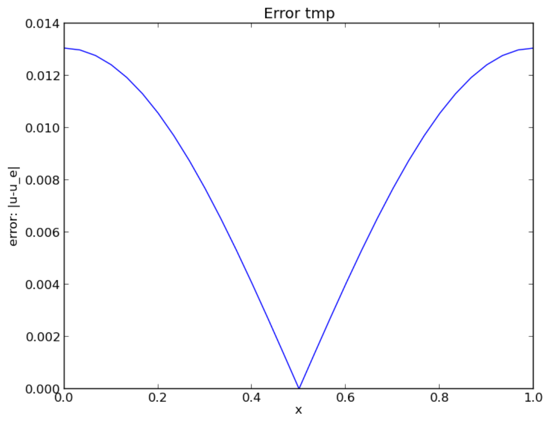
(c) After 3 timesteps



(d) Error after 3 timesteps



(e) After 20 timesteps



(f) Error after 20 timesteps

Figure 1: A first verification of the program. Notice that the axes of the error changes.

We can see with the naked eye, that the numerically found solution differ from the exact solution, and that this difference grows. The shape of the plottet error stays the same during the experiments, but notice that the axes changes. We se that the error is larger at the 20th timestep than at the first timestep.

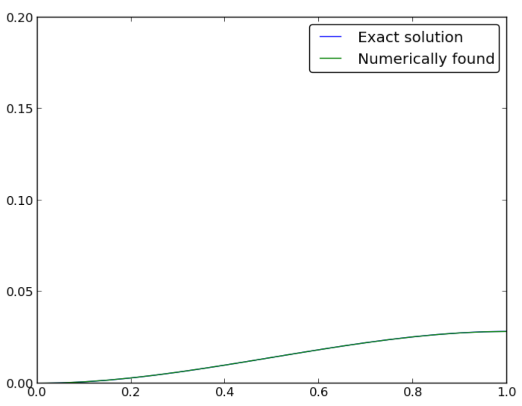
e)

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

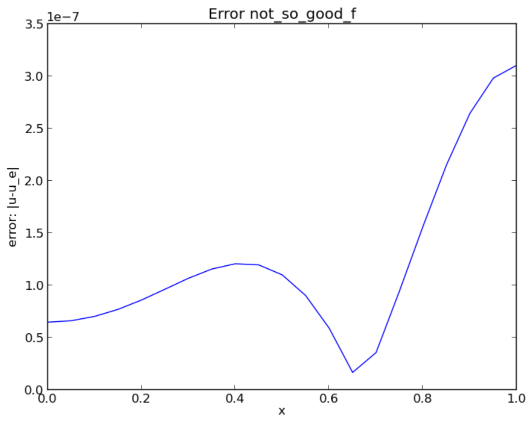
$$u(x,t) = t\int_0^x q(1-q)dq = tx^2(\frac{1}{2} - \frac{x}{3}) \tag{5}$$

and $\alpha(u) = 1 + u^2$. *In the exercise a sympy session is included, which computes an f(x,t) such that the above u is a solution of the PDE problem.*

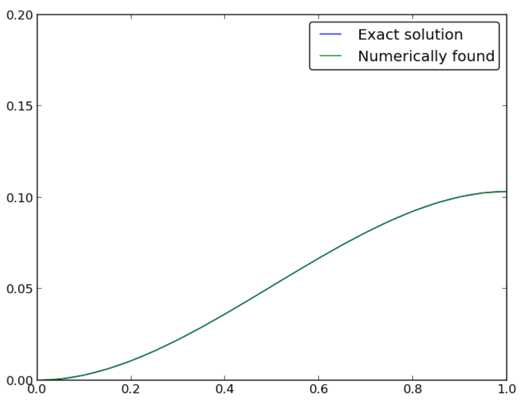
The implementation of this manufactured solution is in the class Manufactured in picard.py. To visualize the result I make a plot at every time-step (we get an animation) of the exact solution and the solution made by the FEniCS-solver. The results are quite good. After 100 timesteps you can not tell the numerically found solution apart from the exact solution, see figure 2. When we look at the error (same figure), we see that after 17 timesteps the error is very small (in the order of 10^{-7}). After 100 timesteps the error is about 50-100 times larger than for timestep 17 (about 10^{-5}).



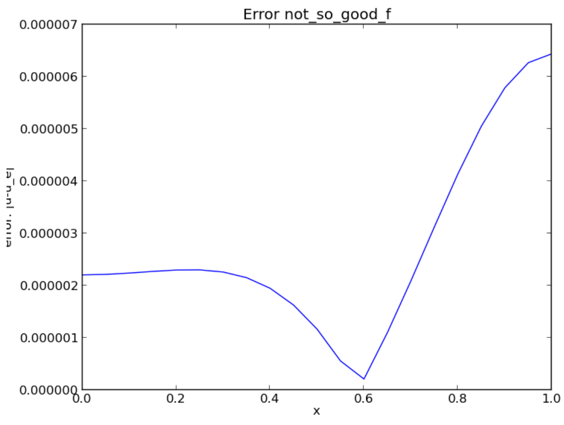
(a) After 17 timesteps



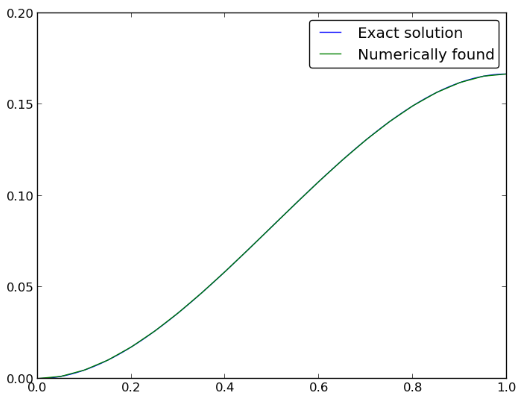
(b) Error after 17 timesteps



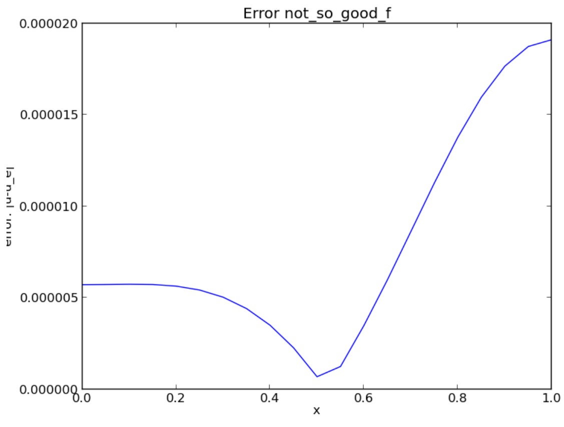
(c) After 62 timesteps



(d) Error after 62 timesteps



(e) After 100 timesteps



(f) Error after 100 timesteps

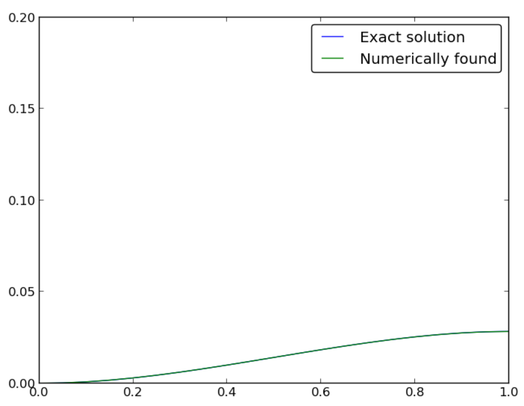
Figure 2: A manufactured solution with one picard iteration.

f)

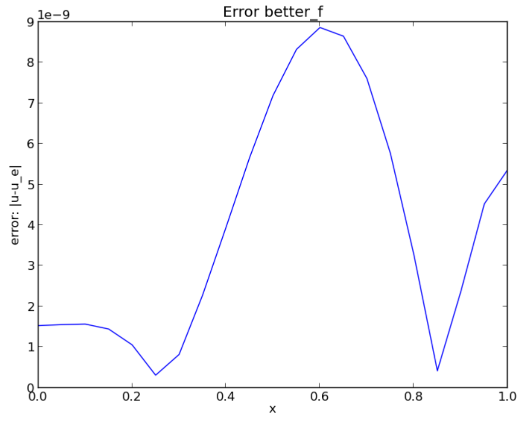
List the different sources of numerical errors in the FEniCS program
TODO: FINN FEILKILDER

g)

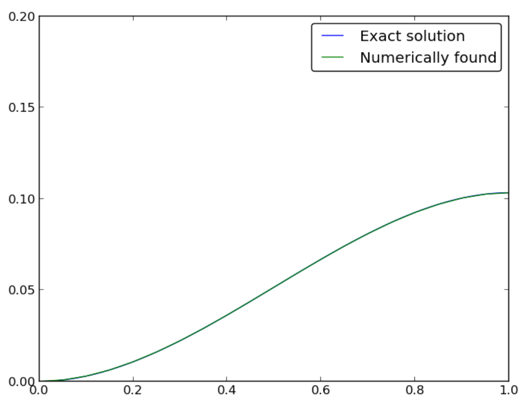
To verify the nonlinear PDE implementation in FEniCS, we can eliminate the error due to a single Picard iteration by computing a manufactured solution corresponding to $\alpha(u_1)$ *, where* u_1 *is the finite element function representation of the solution at the previous time step.* All we have to do to implement this is to rewrite $f(x,t)$. This is done in the exercise text in a sympy session as in exercise e). In the Manufactured class I added another method called make_and_solve_2, which uses the new f during computations. The solution and the error is plotted in figure 3, at the same timesteps as in exercise e). Again there is not possible to tell the exact solution apart from the numerically found solution. The error at the final timestep (n=100) is 1/4 of the error in the final timestep with the old f.



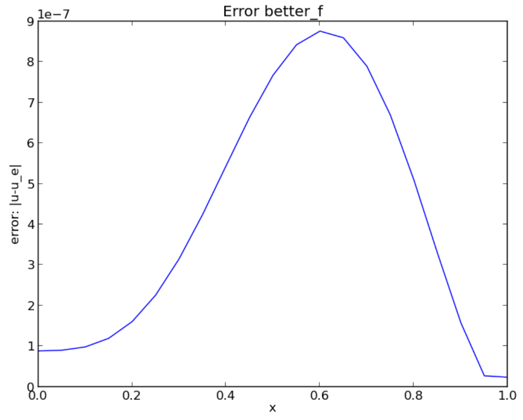
(a) After 17 timesteps



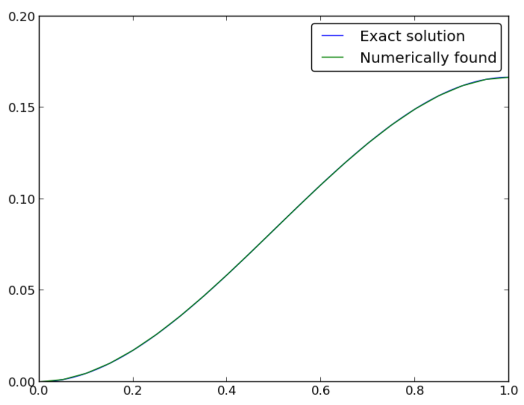
(b) Error after 17 timesteps



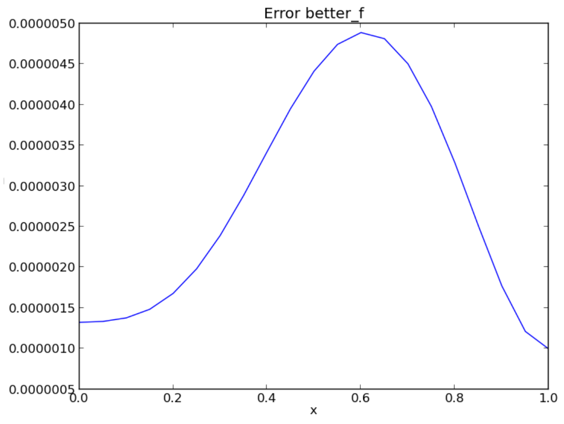
(c) After 62 timesteps



(d) Error after 62 timesteps



(e) After 100 timesteps



(f) Error after 100 timesteps

Figure 3: A manufactured solution with one picard iteration, with a new f.

A convergence rate test was performed, and the results can be found in table 2. We see that the fraction E/h where $h = dt$ stays constant around 0.02 for every h. That means that the error is first order in time, and second order in space, like we expected.

Table 2: Table showing the results of the convergence rate study for the manufactured solution.

E/h	h
0.0318	0.2500
0.0237	0.1111
0.0213	0.0625
0.0189	0.0100
0.0190	0.0025
0.0190	0.0011
0.0190	0.0006

h)

Simulate the nonlinear diffusion of a Gaussian function.
We start out with a system in a Gaussian function. We set the initial condition to

$$I(x, y) = \exp\left(-\frac{1}{2\sigma^2}\left(x^2 + y^2\right)\right) \tag{6}$$

, and the function $f(x, y) = 0$. The nonlinearity function is chosen to be $\alpha(u) = 1 + \beta u^2$, where β is a constantwe can change in different experiments. The implementation of the Gaussian simulation can be found in nonlinear_diffusion_solver.py in the class Gauss. The user can choose whether she wants to look at the 1D- or 2D-case.
I tried a few different values for β, and found that a large β (typically 1000) will get the system to equilibrium faster than a small β (typically 1). When I used β smaller than 1, the effect was not noticeable. Figure 4 shows the 2D case where we start out with a gaussian with s=0.1. We see that the top quickly falls down, and the system reaches equilibrium after about 20 timesteps. After just 5-6 timesteps the top is very close to the "ground". If we start out with a wider gaussian, the top does not fall down to the ground as quickly. In figure 5 is the initial condition a gaussian with s=0.5. These two sets of pictures are part of two movies you can look at. They are called Gauss_movie_s=0,1.gif and Gauss_movie_s=0,5.gif. Enjoy!

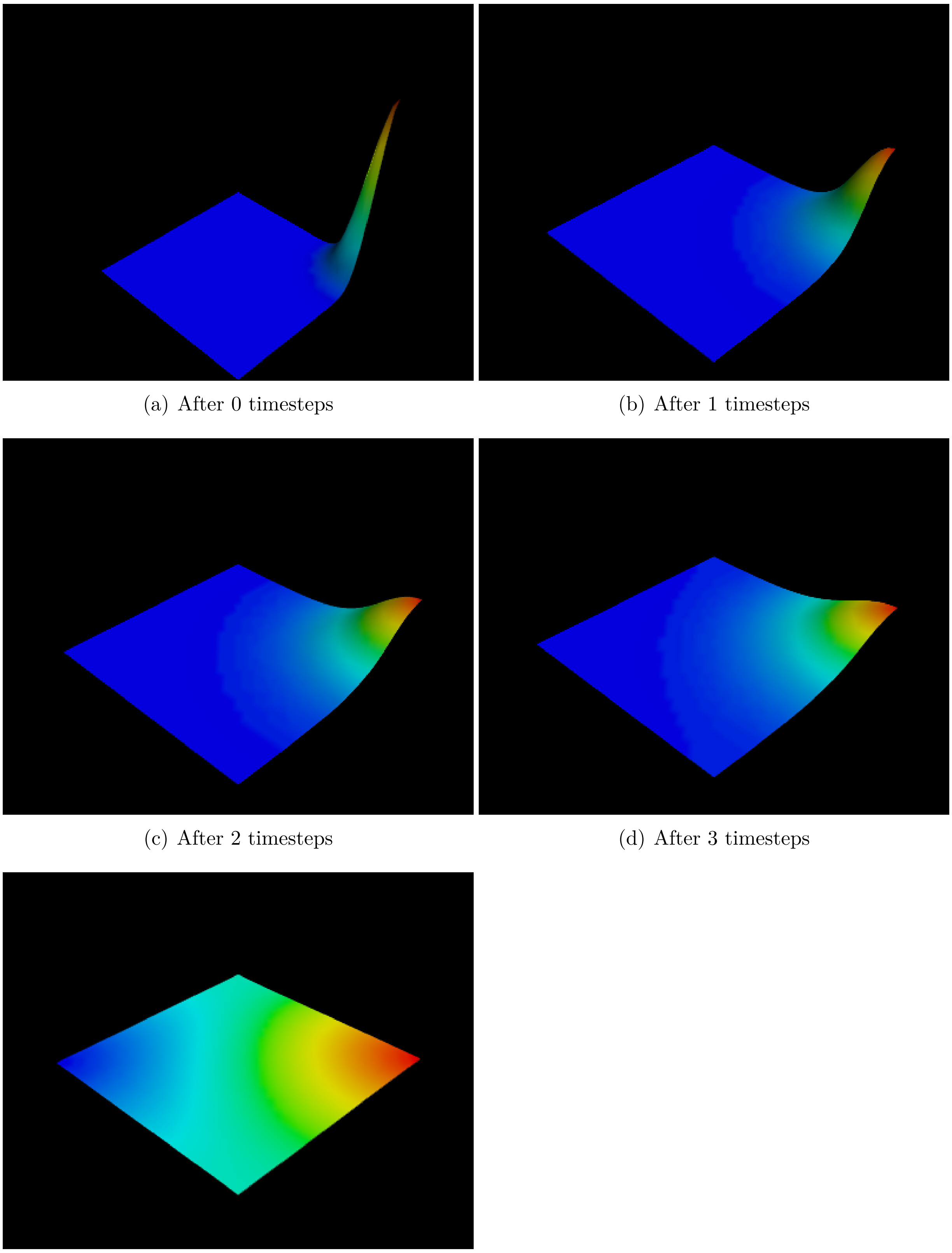

(a) After 0 timesteps
(b) After 1 timesteps
(c) After 2 timesteps
(d) After 3 timesteps
(e) After 20 timesteps

Figure 4: The 2D diffusion problem with a gaussian as initial condition. s= 0.1

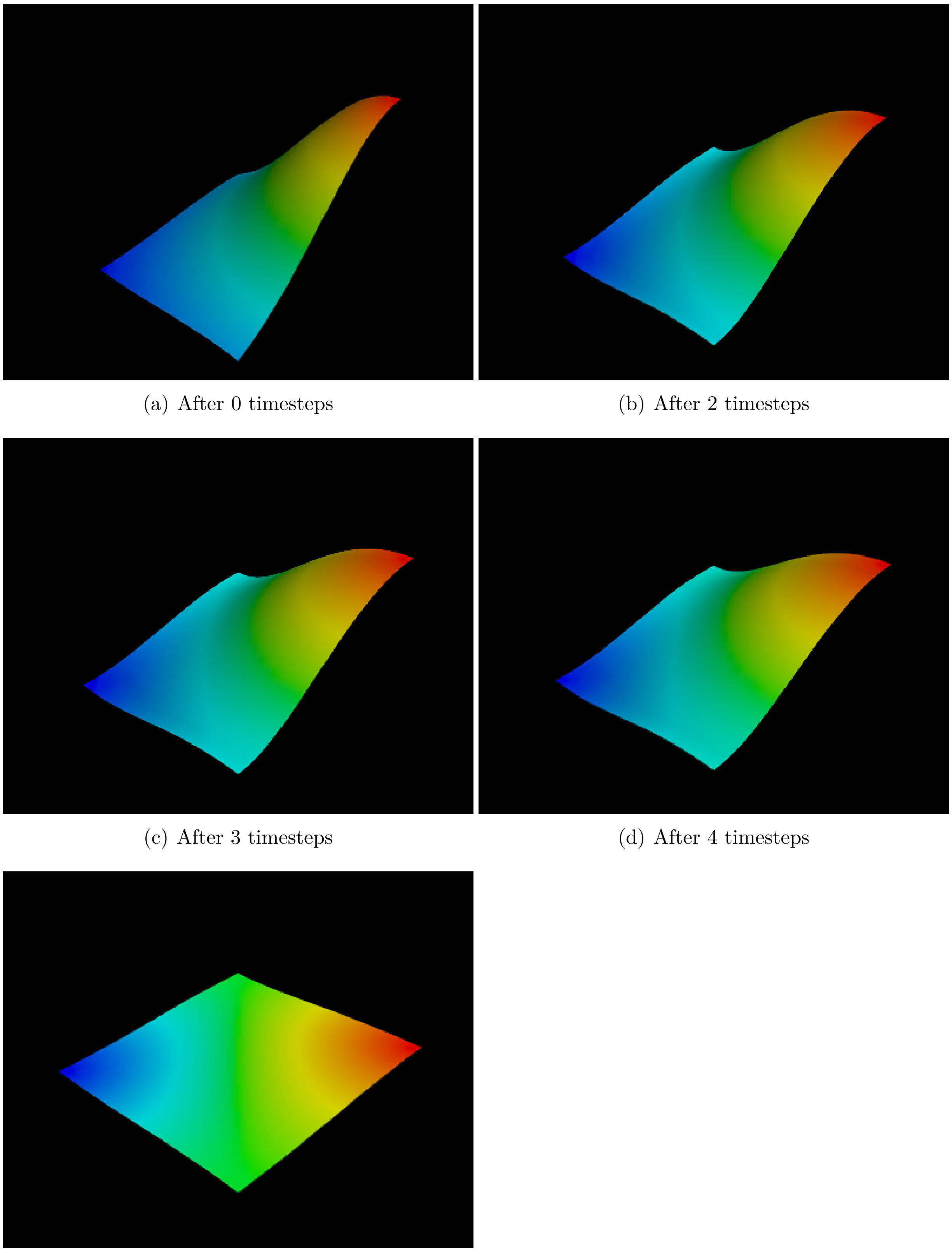

(a) After 0 timesteps (b) After 2 timesteps (c) After 3 timesteps (d) After 4 timesteps (e) After 20 timesteps

Figure 5: The 2D diffusion problem with a gaussian as initial condition. s= 0.5

i)

Formulate a group finite element method for the $\alpha(u)$ coefficient and derive formulas for equation number i in the nonlinear algebraic system to be solved.

The group finite element method is a way of dealing with nonlinear PDE's. The general idea is to approximate the functions of u, with a linear combination of the basis functions:

$$\alpha(u^n) \approx \alpha\left(\sum_j u_j^n \phi_j\right) \approx \left(\sum_j \alpha(u_j^n)\phi_j\right)$$

Let's insert this into the variational form we found in exercise a), and setting $v = \phi_i$. The $L(v)$-function does not contain any function of u, so here we do it the usual way.

$$L(v) = \int_\Omega u^{n-1} v dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n) v dx^3 \approx \int_\Omega \sum_j u_j^{n-1}\phi_j\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n)\phi_i dx^3$$
$$= \sum_j u_j \int_\Omega \phi_i\phi_j dx^3 + \frac{\Delta t}{\rho}\int_\Omega f(x,t^n)\phi_i dx^3 = \sum_j u_j A_{ij} + \frac{\Delta t}{\rho} b_i^n$$

For every $i \in 0, 1, 2, 3, ..., N$.
The whole set of formulas for L(v) (for different i) can be written by the aid of matrices

$$L(v) \approx Au^{n-1} + \frac{\Delta t}{\rho} b^n \tag{7}$$

where $A_{ij} = \int_\Omega \phi_i\phi_j dx^3$, and $b_i^n = \int_\Omega f(x,t^n)\phi_i dx^3$.
Now we will look at the function a(u,v), and use the group finite element approximation.

$$\begin{aligned}
a(u^n, v) &= \int_\Omega u^n v dx^3 + \frac{\Delta t}{\rho}\int_\Omega \alpha(u^n)\nabla u^n \cdot \nabla v dx^3 \\
&\approx \int_\Omega \sum_j u_j^n \phi_j\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega \alpha(\sum_j u_j^n\phi_j)\left(\nabla \sum_k u_k^n\phi_k \cdot \nabla\phi_i\right) dx^3 \\
&= \sum_j u_j^n A_{ij} + \frac{\Delta t}{\rho}\int_\Omega \sum_k u_k^n \alpha(\sum_j u_j^n\phi_j)\left(\nabla\phi_k \cdot \nabla\phi_i\right) dx^3 \\
&= \sum_j u_j^n A_{ij} + \frac{\Delta t}{\rho}\sum_k u_k^n \int_\Omega \sum_j \alpha(u_j^n)\phi_j\left(\nabla\phi_k \cdot \nabla\phi_i\right) dx^3 \\
&= \sum_j u_j^n A_{ij} + \frac{\Delta t}{\rho}\sum_k u_k^n \int_\Omega \sum_j \alpha(u_j^n)\phi_j\left(\nabla\phi_k \cdot \nabla\phi_i\right) dx^3 \\
&= \sum_j u_j^n A_{ij} + \frac{\Delta t}{\rho}\sum_k u_k^n \sum_j \alpha(u_j^n)\left(\int_\Omega \phi_j\left(\nabla\phi_k \cdot \nabla\phi_i\right) dx^3\right) \\
&= \sum_j u_j^n A_{ij} + \frac{\Delta t}{\rho}\sum_k u_k^n \sum_j \alpha(u_j^n) M_{ij}^k
\end{aligned}$$

for every $i \in 1, 2, 3, 4, ..., N$.
We can write these formulas with matrices:

$$a(u^n, v) = Au^n + \frac{\Delta t}{\rho}\sum_k u_k^n M^k \alpha(u^n)$$

where $M_{ij}^k = \int_\Omega \phi_j \nabla\phi_k \cdot \nabla\phi_i dx^3$ and $\alpha(u^n)_i = \alpha(u_i^n)$. The group finite element method for our problem:

$$a(u^n, v) = L(v)$$
$$Au^n + \frac{\Delta t}{\rho}\sum_k u_k^n M^k \alpha(u^n) = Au^{n-1} + \frac{\Delta t}{\rho} b^n$$

The M^k matrices will have a lot of zero-elements. For a given k, we will have three nonzero diagonal elements: $M_k k^k$, $M^k_{k+1,k+1}$ and $M^k_{k-1,k-1}$ (the rest of the diagonal will be zero for this given k). The other nonzero element for this k: $M^k_{k+1,k}$, $M_{k-1,k}$, $M_{k,k-1}$ and $M^k_{k,k+1}$. When we sum these matrices together (running through the possible k-values) we get a tridiagonal matrix.

$$\sum_k u^n_k M^k = T \tag{8}$$

Where T is a tridiagonal matrix buildt up by finite element assembly. The integrals in M^k can be calculated by numerical integration, possibly the Trapizoidal rule if we are using P1 elements.
The equation to be solved:

$$Au^n + \frac{\Delta t}{\rho} T\alpha(u^n) = Au^{n-1} + \frac{\Delta t}{\rho} b^n$$

j)

A Newton method

Newtons method is used when finding points where a function, say $F(U_1, U_2, U_3)$, is zero. For multivariable functions the method looks like this:

$$\sum_{j=1}^{N} \frac{\partial}{\partial U_j} F(U_1^k, ..., U_N^k)\delta U_j = -F(U_1^k, ..., U_N^k) \tag{9}$$

The index called k is an iteration index. We start with an initial guess $U^0 = (U_1^0, U_2^0, ..., U_N^0)$ where we believe the function is close to zero. We then solve eq. (9) to find δU_j, for this initial guess. Our next guess for a zero point is then $U_j^1 = U_j^k + \omega \delta U_j$, for $j = 1, ..., N$. We repeat the process until our function is as close to zero as desired.
We can use the Newton method for our equation. We find the multivariable functions that should be zero.
Our equation now looks like

$$\begin{aligned} a(u^n, v) &= L(v) \\ F(u^n) = a(u^n, v) - L(v) &= 0 \end{aligned} \tag{10}$$

The latter equation should hold for all v, which is spanned by our basis functions ϕ_i. That means it should hold for all the basis functions, and we get a set of functions we want to set to zero:

$$F_i(u^n) = a(u^n, \phi_i) - L(\phi_i) = 0$$

Now we insert $u^n = \sum_{j=1}^{N} U_j^n \phi_j$

$$F_i(U_1^n, U_2^n, ..., U_N^n) = a(\sum_{j=1}^{N} U_j^n \phi_j, \phi_i) - L(\phi_i) = 0 \tag{11}$$

This is a multivariable function, where we could apply Newtons method (eq. (9). The index n tells us at what timestep we're at. It should not be confused with the iteration index k. The next step is to find expressions for the elements in the Jacobian matrix $\frac{\partial F_i(U_1^{n,k},...,U_N^{n,k}}{\partial U_j^n}$. Start by finding the derivatives of $a(u^n, \phi_i)$

$$\frac{\partial a(\sum_{j=1}^N U_j^{n,k}\phi_j, \phi_i)}{\partial U_j^n} = \frac{\partial}{\partial U_j^n}\left(\sum_p^N U_p^n \int_\Omega \phi_p\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega \sum_p^N U_p^n \nabla\phi_p \cdot \nabla\phi_i \alpha(\sum_q^n U_q^n\phi_q)dx^3\right) \tag{12}$$

$$\tag{13}$$

This should be evaluated for $U_j^{n,k}$ of course. The first term is quickly found

$$\frac{\partial}{\partial U_j^n}\left(\sum_p^N U_p^n \int_\Omega \phi_p\phi_i dx^3\right) = \int_\Omega \phi_j\phi_i dx^3$$

For the second term we need to calculate some derivatives first

$$\frac{\partial}{\partial U_j^n}\left(\sum_p^N U_p^n \nabla\phi_p \cdot \nabla\phi_i\right) = \nabla\phi_j \cdot \phi_i$$

$$\frac{\partial}{\partial U_j^n}\left(\alpha(\sum_p U_p^n\phi_p)\right) = \frac{\partial\alpha(u^n)}{\partial u^n}\frac{\partial u^n}{\partial U_j^n} = \alpha'(u^n)\frac{\partial(\sum_p U_p^n\phi_p)}{\partial U_j^n} = \alpha'(u^n)\phi_j$$

Since we should evaluate the derivative at the guess $u^{n,k} = \sum_j U_j^{n,k}\phi_j$, this derivative becomes

$$\frac{\partial}{\partial U_j^n}\left(\alpha(\sum_p U_p^n\phi_p)\right) = \alpha'(u^{n,k})\phi_j$$

Now we are ready to find the second term. We use the product rule to find the derivative

$$\begin{aligned}\frac{\partial}{\partial U_j^n}\left(\frac{\Delta t}{\rho}\int_\Omega \sum_p^N U_p^n \nabla\phi_p \cdot \nabla\phi_i \alpha(\sum_q^n U_q^n\phi_q)dx^3\right) &= \frac{\Delta t}{\rho}\int_\Omega \left[\nabla\phi_j \cdot \nabla\phi_i\alpha(u^{n,k}) + \sum_p^N U_p^n \nabla\phi_p \cdot \nabla\phi_i \alpha'(u^{n,k})\phi_j\right] \\ &= \frac{\Delta t}{\rho}\int_\Omega \left[\nabla\phi_j \cdot \nabla\phi_i\alpha(u^{n,k}) + \nabla u^{n,k} \cdot \nabla\phi_i\alpha'(u^{n,k})\phi_j\right]dx^3\end{aligned} \tag{14}$$

The total derivative of a:

$$\frac{\partial a(\sum_{j=1}^N U_j^{n,k}\phi_j, \phi_i)}{\partial U_j^n} = \int_\Omega \phi_j\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega \left[\nabla\phi_j \cdot \nabla\phi_i\alpha(u^{n,k}) + \nabla u^{n,k} \cdot \nabla\phi_i\alpha'(u^{n,k})\phi_j\right]dx^3 \tag{15}$$

We need to find the derivative of $L(\phi_i)$. This is very easy because this function does not depend on u^n, so the derivative is zero (it does depend om u^{n-1}, but this is a known function from the last timestep). The equation we need to solve (solve for δU_j) is

$$\begin{aligned}\sum_{j=1}^N \left[\int_\Omega \phi_j\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega \left[\nabla\phi_j \cdot \nabla\phi_i\alpha(u^{n,k}) + \nabla u^{n,k} \cdot \nabla\phi_i\alpha'(u^{n,k})\phi_j\right]dx^3\right]\delta U_j = \\ -\int_\Omega u^{n,k}\phi_i dx^3 + \frac{\Delta t}{\rho}\int_\Omega \nabla u^{n,k} \cdot \nabla\phi_i\alpha(u^{n,k})dx^3 + \int_\Omega u^{n-1}\phi_i + \frac{\Delta t}{\rho}f(x,t^n)\phi_i dx^3\end{aligned}$$

For $i \in 1, 2, 3, ..., N$

k)

Reduce the problem to one space dimension. Derive formulas for the entries in the Jacobian matrix, and the right hand side using P1 elements and the Trapezoidal rule for integration. The Jacobian matrix element for one dimension (x):

$$\frac{\partial F_i(u^{n,k})}{\partial U_j^n} = \int_\Omega \phi_j \phi_i dx + \frac{\Delta t}{\rho} \int_\Omega \phi_j' \phi_i' \alpha(u^{n,k}) + u^{n,k\prime} \phi_i' \frac{\partial \alpha(u^{n,k})}{\partial u} \phi_j dx$$

The right hand side for 1D:

$$-\int_\Omega u^{n,k} \phi_i dx + \frac{\Delta t}{\rho} \int_\Omega u^{n,k\prime} \phi_i' \alpha(u^{n,k}) dx + \int_\Omega u^{n-1} \phi_i + \frac{\Delta t}{\rho} f(x, t^n) \phi_i dx$$

Lets look at the different integrals in the Jacobian matrix

$$\int_0^L \phi_j \phi_i dx = \delta_{ij} \frac{2\Delta x}{3} + \delta_{ij\pm 1} \frac{\Delta x}{6} \tag{16}$$

This is calculated by changing variable to a variable X in the reference element. $x = x_m + \frac{1}{2}\Delta x X$. I did this in the highly recommended project, and I am running out of time so I just write up the answer. The delta-functions tell us that we get a tridiagonal matrix.

$$\begin{aligned} \int_0^L \phi_i' \phi_j' \alpha(\sum_p U_p \phi_p) dx &\approx \frac{L}{N} \sum_k^{N-1} \phi_j'(x_k) \phi_i' \alpha(\sum_p U_p \phi_p(x_k)) \qquad (17) \\ &= \sum_{k=1}^{N-1} \phi_j'(x_k) \phi_i'(x_k) \alpha(u_k^n) \end{aligned}$$

The derivatives:

$$\phi_j'(x_k) = 2/h \begin{cases} 1 & \text{if } k = j, \\ -1 & \text{if } k = j+1. \end{cases} \tag{18}$$

$$\tag{19}$$

The total sum:

$$\sum_{k=1}^{N-1} \phi_j'(x_k) \phi_i'(x_k) \alpha(u_k^n) = 4/h^2 \begin{cases} \alpha(u_i^n) - \alpha(u_{i+1}^n) & \text{if } i = j, \\ -\alpha(u_{i+1}^n) & \text{if } i = j \pm 1. \end{cases} \tag{20}$$

$$\tag{21}$$

We have to do the the other integrals in an equvialent way.