INF5620: FINAL PROJECT

KRISTIAN PEDERSEN, KRISTIAN.BERSETH.PEDERSEN@GMAIL.COM

1. Introduction

My solution of the final project in INF5620. It is somewhat rough on the edges as I have spent litteral days on bugfixing and need the remaining time to practice for the exam.

2. a.)

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

Discretizing in time and using backward euler we get:

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

or, setting $\kappa = \frac{\rho}{\Delta t}$, $u = u^k$ and $u_p = u^{k-1}$:

$$u = u_p + \kappa(\nabla \cdot (\alpha(u)\nabla u) + f(\bar{x}, t^k))$$

Finding the variational form of the problem:

$$\langle u, v\rangle = \langle u_p, v\rangle + \kappa(\langle \nabla \cdot (\alpha(u)\nabla u), v\rangle + \langle f(\bar{x}, t^k), v\rangle)$$

Sorting and applying Green's Identity along with the neumann condition:

$$\langle u_p, v\rangle + \kappa\langle f(\bar{x}, t^k), v\rangle) = \langle u, v\rangle + \kappa\langle \nabla \cdot (\alpha(u)\nabla u), v\rangle$$

And we find

$$a = \langle u, v\rangle + \kappa\langle \nabla \cdot (\alpha(u)\nabla u), v\rangle$$
$$L = \langle u_p + \kappa f(\bar{x}, t^k), v\rangle)$$

3. b.)

Setting $u^q = u_p$ for q = 0 and iterating over q = 1,2,3 ... we get the following systems from picard iteration:

$$-\langle u_p, v\rangle - \kappa\langle f(\bar{x}, t^k), v\rangle) + \langle u^q, v\rangle + \kappa\langle \nabla \cdot (\alpha(u^{q-1})\nabla u^q), v\rangle = 0$$

4. c.)

Inserting for q = 1 and renaming u^1 to u we find:

$$-\langle u_p, v\rangle - \kappa\langle f(\bar{x}, t^k), v\rangle) + \langle u, v\rangle + \kappa\langle \nabla \cdot (\alpha(u_p)\nabla u), v\rangle = 0$$

The implementation of this can be found in the file dummy.py in the Solver class' Picard function.

5. D.)

Implemented the problem is implemented in the generate_first_verification(h) function in the Problem class of dummy.py and the convergance test is launched from the convergance_test1 function in the Tester class. Here I would probably have changed the structure a bit and add a proper test (instead of just printings) if I hadn't been so short on time.

6. E.)

Implemented the problem in the generate_first_verification(h) function in the Problem class of Solver.py and a test running some T values is implemented in the manufactured_test() function in the tester class. The error seems reasonably small, but not non-existant. It seems to scale with T, but that is not unreasonable considering the choice of error measure.

7. F.)

Some factors:

(1) The error from discretization in time
(2) The error from the assumption that the picard method will converge (A guess, but I think one can find some sufficiently ugly $\alpha(u)$s to make it true)
(3) The error from doing a finite amount (one) of picard iterations
(4) Error caused while interpolating I and other functions
(5) Error caused during FEniCS's solution process (Though I am unsure wether or not this is included in point 5 and 7 when FEniCS uses "exact" solvers for linear systems)
(6) Errors introduced by floating point arithmetic.

8. G.)

I postponed this and promptly forgot it :(

9. H.)

Implemented the problem in the generate_gaussian(beta) function in the Problem class of Solver.py and a test running some β values is implemented in the gaussian_test() function in the tester class. I have not implemented a way to plot the solution as I am running low time and the answer is obviously wrong (for one it is invariant in beta). I have spent some time trying to locate the bug, but it is nowhere to be found.

10. I.)

The group finite method is based on the assumption:

$$\alpha(u) = \alpha(\Sigma_{i=1}^{n} \phi_i u_i) \approx \alpha(\Sigma_{i=1}^{n} u_i)\phi_i$$

Then for j = 1,2...n :

$$\langle \alpha(u), \phi_j \rangle = \langle \alpha(\Sigma_{i=1}^{n} \phi_i u_i), \phi_j \rangle \approx \langle \alpha(\Sigma_{i=1}^{n} u_i)\phi_i, \phi_j \rangle = \alpha(\Sigma_{i=1}^{n} u_i)\langle \phi_i, \phi_j \rangle$$

As we're using P1 elements $\langle \phi_i, \phi_j \rangle = 0$ for $j \neq i, i+1$ or $i-1$. Using the standard P1 integrals from the compendium/slides we find:

$$\langle \alpha(u), \phi_j \rangle \approx \frac{h}{6}(\alpha(u_{i-1} + \alpha(u_i) + \alpha(u_{i+1})$$

11. j.)

For i = 1,2..n we can define F_i as:

$$F_i = a(u, \phi_i) - L(\phi_j) = \int_\Omega (u - u_p)\phi_i - \kappa a(u)\nabla u \cdot \nabla \phi_i + f(\bar{x}, t^k)dx = 0$$

Defining the Jacobian as:

$$J_{i,j} = \frac{\partial F_i}{diu_j}$$

We can use the fact that L and ϕ_i is constant with respect to u_j and apply the chain rule to the second term to get:

$$J_{i,j} = \int_\Omega \frac{\partial u}{\partial u_j}\phi_i + \kappa \frac{\partial a(u)}{\partial u_j}\nabla u \cdot \nabla \phi_i + a(u)\nabla \frac{\partial u}{\partial u_j} \cdot \nabla \phi_i dx$$

Inserting the galerkin expansion of u: $u = \Sigma_{i=1}^{n} u_i)\phi_i$ and noticing that the derivation will remove all the terms but the u_j term we find:

$$J_{i,j} = \int_\Omega \phi_j \phi_i + \kappa a'(u)\phi_j \nabla u \cdot \nabla \phi_i + a(u)\nabla \phi_j \cdot \nabla \phi_i dx$$

Where we used the core rule to simplify the derivative of a(u)

12. k.)

Inserting for one dimension we get:

$$J_{i,j} = \int_\Omega \phi_j \phi_i + \kappa a'(u)\phi_j \frac{\partial u}{\partial x}\phi_i'(x) + a(u)\phi_j' \phi_i' dx$$

After several attempts to get the algebra right I have given up on the rest of this exercise. I tried using the simplicity of P1 integrals and collapsing sums when using trapezodial integration. The expression still seemed extremely complex. Maybe I've made a mistake in the above calculations?

13. l.)

I am assuming we would find that the expression from k to be similar to a finite element approximation, as often is the case when using P1 elements and the trapezodial rule. Hard to say which without the exact expression though.