Chapter 1

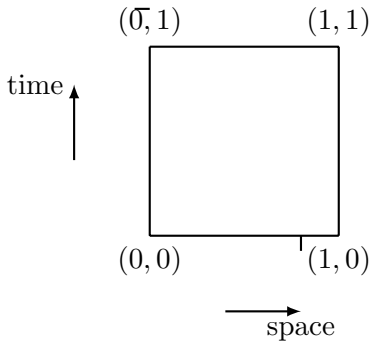
Implementation and Numerical Results

1.1 General

In this chapter we will look at how the *solver* we derived actually performs in some test cases. We looked at problems with a one-dimensional space domain $\mathcal{S} = (0, S)$ and a time-interval $\mathcal{T} = (0, T)$, that is

$$\Omega = (0, S) \times (0, T), \quad S, T > 0 \tag{1.1}$$

and which we discretised using uniformly-sized rectangular elements. Let N_x be the number of elements in space and N_t the number of elements in time. As a finite-dimensional approximation space we used piecewise linear polynomials in space and time for the approximation of σ_h as well as u_h and chose the local basis



$$\begin{aligned} \phi_1(x,t) &= (1-x) \cdot(1-t) \\ \phi_2(x,t) &= x \cdot(1-t) \\ \phi_3(x,t) &= (1-x) \cdot t \\ \phi_4(x,t) &= x \cdot t \end{aligned} \tag{1.2}$$

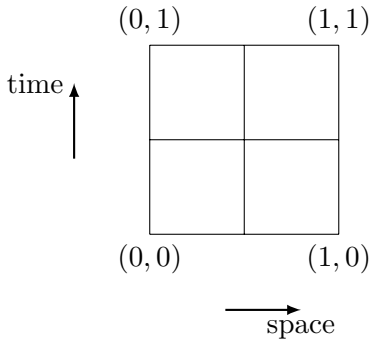
We arranged the grid points the same way the local basis is labeled, that is for a fixed t_j all elements in space with $x_i < x_{i+1}$, and subsequently all space elements for t_{j+1}. As previously mentioned the we arranged σ_h and u_h such that they are two concatenated vectors. Hence one obtains a system of equations with $m = (N_x+1)\cdot(N_t+1)$ degrees of freedom for σ_h and the same again for u_h. The solution vector is therefore of $2m$ while the matrices have a size of $2m \times 2m$. It is possible to use this local basis for σ_h and u_h because we are in the one-dimensional case and

therefore $\nabla u = \partial_x u = \sigma$ and $div(\sigma) = \partial_x \sigma$. For $dim(\mathcal{S}) > 1$ we would have to treat this problem differently using for example Raviart-Thomas elements [source], which is however beyond the scope of this thesis *but could be of interest in future investigations.* In order to compute the individual integral terms we used a quadrature rule of degree three, that is we get the exact integral for the linear apprximations of the functions. The admissible boundary conditions are a mixture of Dirichlet and Neumann type and are directly imposed in the system, either in u or σ, respectively.

1.2 Multigrid Details

We implemented a geometric multigrid V-cycle, where the number of elements on the coarse grid and the number of levels are chosen and then all other grids, interpolation operators, and finer level operators are generated automatically. The reason behind this is to guarantee nested meshes. As discussed previously there is no unique best problem indepedent coarsening strategy, and especially in the case of space-time discretisations it has a great effect on the overall performance. Unfortunately the literature on what could be a favourable strategy in a least squares space-time set up was very sparse and therefore we decided to use a classical space-time coarsening approach, where the size of each rectangle grows by a factor of 4 with each coarsening because the length of the element is doubles in each direction, see figure [...] below. This seemed like a reasonable approach, since we are dealing with a coupled first order system, and hence considerations like in [**?**], that *showed* it would to be preferrable to keep the quotient $\lambda = \frac{d\delta t}{\delta x^2}$ (where d decribes the diffusion term and δt and δx the respective step sizes) close to one on all levels, did not really apply here because we have no second order derivative, that would lead to the square term in space. So under the assumption that the diffusion term d is not too different from one and we choose a similar stepsize in space and time, the standard space-time coarsening would lead to a coefficient that is the same on all levels as well as close to one. More extensive testing on this beyond the scope of this thesis. *Rough estimates showed that for the linear case this seemed to be the case as a interpolation operators of type [**?**] seemed to lead to slower convergence.*

We then chose to use interpolation weights according to the following scheme



SMOOTHER

The Hessian that arises from the minimisation problem is not simply the discretisation of one differential operator as one would for example obtain when solving a Poisson equation. Instead it is a combination of the Laplacian operator in space, the first order advection in time and the linearised (and adapted) first and second order derivatives of $f(u)$.

Therefore it is complicated/not really possibly to relate to physical things .... ?
What is the idea?

The Newton iteration at hand looks as follows:

$$y_{k+1} = y_k - H_J^{-1}(y_k)(\nabla J(y_k)) \tag{1.3}$$

where $y_k = (\sigma_k, u_k)^T$ and the hessian H_J is defined as $H_J(y_k) = D^2 J(\sigma_k, u_k)$.
Written in matrix notation one obtains the following system which is split up in the linear and non-linear part:

$$\begin{aligned} \nabla J(y_k) = &\begin{bmatrix} -\langle u_t, div(\tau)\rangle + \langle \sigma_k, \tau\rangle + \langle div(\sigma_k), div(\tau) - \langle \nabla u_k, \tau\rangle \\ \langle u_t, v_t\rangle - \langle v_t, div(\sigma_k) + \langle \nabla u, \nabla v\rangle - \langle \sigma_k, \nabla v\rangle \end{bmatrix} \\ + &\begin{bmatrix} +\langle f(u_k), div(\tau)\rangle \\ \langle div(\sigma_k), f'(u_k)v\rangle + \langle f(u_k), f'(u_k)v\rangle - \langle (u_k)_t, f'(u_k)v\rangle - \langle v_t, f(u)\rangle \end{bmatrix} \end{aligned} \tag{1.4}$$

$$\begin{aligned} H_J(y_k) = &\begin{bmatrix} \langle \tau, \rho\rangle + \langle div(\tau), div(\rho)\rangle & -\langle \rho, \nabla v\rangle - \langle v_t, div(\rho)\rangle \\ -\langle \tau, \nabla w\rangle - \langle w_t, div(\tau) & \langle v_t, v_t\rangle + \langle \nabla v, \nabla w\rangle \end{bmatrix} \\ + &\begin{bmatrix} 0 & -\langle div(\tau), f'(u_k)\cdot w\rangle \\ -\langle div(\rho), f'(u_k)v\rangle & -\langle (u_k)_t, w^T f''(u_k)v\rangle + \langle div(\sigma_k), w^T f''(u_k)v\rangle + \langle f'(u_k)w, f'(u_k)w\rangle + ... \\ & ... + \langle f(u_k), w^T f''(u_k)v\rangle - \langle w_t, f'(u_k)v\rangle - \langle v_t, f'(u_k)w\rangle \end{bmatrix} \end{aligned} \tag{1.5}$$

Using basis and test functions of the type .... P1. Approximate the integrals using Gaussian quadrature of degree three, therfore computing integrals of polynomials of degree one, which includes our basis functions, however not exact for f which is represented as ... The system one obtains from the first
u as a sum of basis functions, σ as well

reorganise these terms to set up the Newton iteration how does my newton iteration look like exactly?
write that up ....
put in additional terms for equation, for now u_p is u from previous iteration. 3rd line will go on rhs, last line as well, 2nd line on the left

$$\begin{bmatrix} A_{\sigma\sigma} & \\ & A_{uu} \end{bmatrix} \cdot \begin{bmatrix} \sigma \\ u \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ diag(M_t \times G_h \cdot (f'(u)\cdot v)) & -diag(G_t \times M_h)\cdot \end{bmatrix} = \begin{bmatrix} -\langle div(\tau), f(u_p)\rangle \\ \langle v_t, f(u_p)\rangle \end{bmatrix}$$

Then leave it at the continuous problem formulation though.
Next thing that we need is the space-time discretisation.
what changes between Dirichlet and Neumann boundary conditions?
Boundary conditions are directly enforced. How to get symmetry back?
Some sort of convergence test?
Eventually come to multigrid implementation.
The chosen basis function are from Q_1, that is the space of For each basis function we have that and

$$\phi_{ij}(x_k, t_l) = \begin{cases} 1 & \text{if } (k,l) = (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

how to make this better ....? line search requires gradient evaluations in inner loop

1.3 Heat Eqn, LSFEM, space - time parallel, zero forcing term

1.3.1 Basic Implementation

block structure of σ and u. First all σ then all u. Maybe in future work, couple them.

1.3.2 Smoother

PARALLELISATION IS KEY – CHOSE SMOOTHERS ACCORDINGLY
Smoothers are a key part of an efficient multigrid algorithm. And while many of them may theoretically be a possibility, in practice the choice of a suitable smoother is often not an easy one. We would like that the combination of coarse grid correction and smoother reduces the error efficiently for all frequencies. In the case of a geometric multigrid applied to an elliptic problem the coarse grid correction captures the low frequency error quickly while most smoothers like (block-) Jacobi or Gauss-Seidel reduce the high frequency error well and therefore the two are a favourable synthesis. In our set up the situation is more difficult. Due to the separated structure of the two variables σ and u, but there *grid* and contentual connection it might be favorable to use a block smoother that takes this relationship into account. We therefore constructed a smoother for which degrees of freedom which are connected on the grid can be chosen and then the corresponding σ and u variables will be treated together. That is we will obtain the following ....

put little graphics thing?

tested on : PUT PICTURE OF SOLUTION AFTER 5,10, ... SMOOTHING STEPS

for the particular problem one is looking

1.3.3 Coarsening Strategies

talk about λ - coefficient and paper, different operators, see if it changes anything, graphs, convergence tests, etc.

1.4 Monodomain Equation

- general construction
- talk about non-convexity
- then about nonlinear solvers