

# Chapter 1

## Implementation and Numerical Results

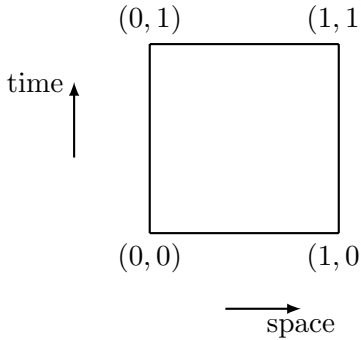
### 1.1 Discretisation Scheme

*in this chapter we will only be talking about approximate solutions from the finite dimensional subspaces and therefore in order to simplify notation refer to what was before called  $u_h$  or  $\sigma_h$  as  $u$  and  $\sigma$ .*

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

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

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 bilinear polynomials  $Q_1$  in space and time for the approximation of  $\sigma_h$  as well as  $u_h$  and chose the local basis



$$\begin{aligned} \phi_{11}(x, t) &= (1 - x) \cdot (1 - t) \\ \phi_{12}(x, t) &= x \cdot (1 - t) \\ \phi_{21}(x, t) &= (1 - x) \cdot t \\ \phi_{22}(x, t) &= x \cdot t \end{aligned} \quad (1.2)$$

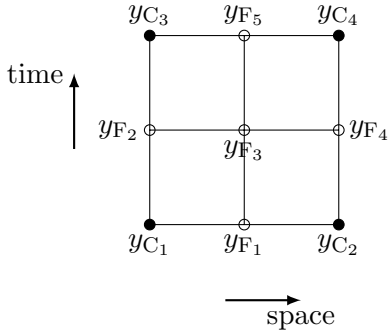
If we translate this from local to the global coordinates and scale appropriately according to the mesh size with an appropriate projection, we have that

$$\phi_{ij}(x_k, t_l) = \begin{cases} 1 & \text{if } (k, l) = (i, j) \\ 0 & \text{otherwise} \end{cases} \quad (1.3)$$

We arrange 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  $\sigma_h$  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  $\sigma_h$  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  $\sigma_h$  and  $u_h$  because we are in the one-dimensional case and therefore  $\nabla u = \partial_x u = \sigma$  and  $\text{div}(\sigma) = \partial_x \sigma$ . 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 approximations 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  $\sigma$ , respectively.

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 independent 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. It means that with each coarsening step the number of elements reduces by a factor of  $(2^{-2})$  and therefore also the degrees of freedom. 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 be preferable to keep the quotient  $\lambda = \frac{d\Delta t}{\Delta x^2}$  (where  $d$  describes the diffusion constant and  $\Delta t$  and  $\Delta x$  the respective step sizes) close to one on all levels, did not 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. In section (6.3.2) we will see some numerical test cases for this. *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, where filled dots represent coarse grid points and the blank circles fine grid points



$$\begin{aligned}
 s(y_{C_i}) &= s(y_{C_i}) \quad \forall i \\
 s(y_{F_1}) &= \frac{1}{2}y_{C_1} + \frac{1}{2}y_{C_2} \\
 s(y_{F_2}) &= \frac{1}{2}y_{C_1} + \frac{1}{2}y_{C_3} \\
 s(y_{F_3}) &= \frac{1}{4}y_{C_1} + \frac{1}{4}y_{C_2} + \frac{1}{4}y_{C_3} + \frac{1}{4}y_{C_4} \\
 s(y_{F_4}) &= \frac{1}{2}y_{C_2} + \frac{1}{2}y_{C_4} \\
 s(y_{F_5}) &= \frac{1}{2}y_{C_3} + \frac{1}{2}y_{C_4}
 \end{aligned} \tag{1.4}$$

They are constructed respectively to go recursively from one level to the next, and this is for either  $\sigma_h$  or  $u_h$ . If  $\tilde{I}$  is an interpolation matrix of the above type then we will have that the overall interpolation operator will have the following form, that is we interpolate independently for the two variables.

$$I = \begin{bmatrix} \tilde{I} & 0 \\ 0 & \tilde{I} \end{bmatrix} \tag{1.5}$$

The interpolation matrix from level  $k-1$  to level  $k$  is as before denoted by  $I_{k-1}^k$  and we have that  $I_{k-1}^k \in \mathbb{R}^{2m_k \times 2m_{k-1}}$ , where  $m_k$  and  $m_{k-1}$  denotes the number of points on the space time grid on the respective level.

Visually the function on the right really appears to be a good interpolation of the function on the left, transmitting *global* information. *In the multigrid implementation we remove the influence of the boundary conditions.*

Figure 1.1: Interpolation of the solution to the heat equation with  $u(x, 0) = \max(1 - 2x, 0)$ ,  $D(x) = 0.1$  and homogeneous Neumann boundary conditions in time on a unit square from  $8 \times 8$  to  $16 \times 16$  elements. The interpolated solution of  $\sigma$  is not shown here as  $\tilde{I}$  simply applied to  $\sigma$  and  $u$  independently.

### 1.2.1 Smoothers

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. But as we are also trying to develop a highly parallelisable solver it is of course important to also be taking that into account when choosing a smoother, therefore a regular Gauss-Seidel iteration is for example not a suitable choice as it works sequentially. In our set up we also have the additional feature of the two coupled variables  $\sigma_h$  and  $u_h$ , which are separated in the sense that they are two concatenated vectors but have the contentual connection. Therefore it might be favorable to use a (block-) smoother that takes this relationship into account. Hence we decided to use a Vanka type [source?] block Jacobi relaxation. That is a block Jacobi smoother where each block contains all degrees of freedom associated to one grid point in the space-time domain. That is we choose rectangular patches of size  $p \times q$  on the domain, that is  $p$  points in space and  $q$  in time, construct a submatrix containing all entries of  $A$  associated to those nodes for  $\sigma_h$  and  $u_h$  and all of the corresponding coupling terms and directly invert the small submatrix. We partition the entire domain like that, while potentially having smaller patches on the boundary, and finally assemble a preconditioner  $P$  containing the inverted submatrices. Due to the organisation of  $\sigma_h$  and  $u_h$ ,  $P$  will not be a block diagonal matrix. For  $p = q = 2$  we would have the following patches, where  $\tilde{C}_i$  describes the set of indices associated with the patch, and  $C_i$  the corresponding submatrix, which are each of size  $8 \times 8$ .

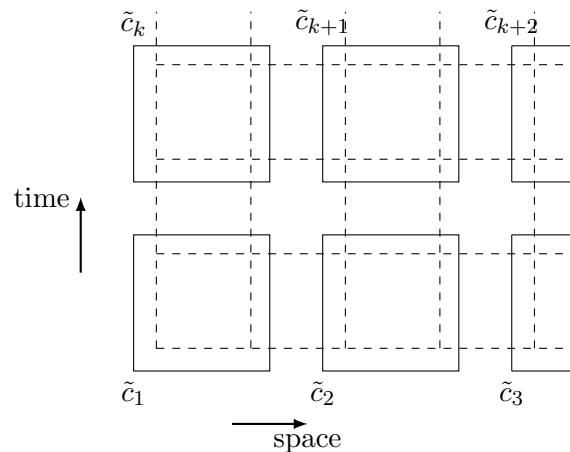


Figure 1.2: Schematic overview of the construction of the blocks of damped Jacobi smoother

#### *damped Jacobi*

Below we can see how the smoother acts on a random initial guess for the linear system  $As = f$ , where  $A$  is the least squares finite element matrix arising from the functional  $J([\sigma, u], 0)$  with

the discretisation chosen as described in chapter 4 and the previous parts of this chapter, and  $f$  is a zero right-hand side that only contains the boundary conditions. That is we look at the iteration

$$e_{k+1} = e_k + \omega P(f - Ae_k) \quad (1.6)$$

where  $P$  is the preconditioner and  $\omega$  stands for the damping factor. Since we have a zero forcing term, the exact solution is zero everywhere if we have homogenous boundary conditions, and hence  $e_k$  describes the remaining error in each iteration starting from an initial guess.

Figure 1.3: Iterates after  $k = 0, 3, 8, 20$  iterations for a patch size of  $1 \times 1$ , and a grid with  $11 \times 11$  elements. needed more than 9000 iterations in order for the norm of the residual to be  $< 10^{-9}$ .

We can see that the damped block Jacobi smoother really leads to a fast reduction of the high frequency error whereas the low frequency error is only reduced very slowly. *something about computationally very cheap*. Again we only plotted  $u$  as  $\sigma$  behaves in the same way, simply with different boundary conditions.

Figure 1.4: Iterates after  $k = 0, 3, 8, 20$  iterations for a patch size of  $3 \times 4$ , and a grid with  $11 \times 11$  elements. needed more than 3177 iterations in order for the norm of the residual to be  $< 10^{-9}$ .

Another type of smoother that we tested is a Gauss-Seidel *line smoother* [?] which can also be categorised as a Vanka smoother *write more about that reference?*. One considers each spatial degree of freedom individually but together for all times as well as the values for  $\sigma$  and  $u$ . We extract all matrix values for each of the  $x_i$ , to construct submatrices. We then first solve exactly for all odd subsets of  $\tilde{c}_i$ , here coloured in black and update the solution before solving for all even subsets, here coloured in red. Hence for all subsets labeled in black we can solve in parallel, and subsequently for all subsets in red.

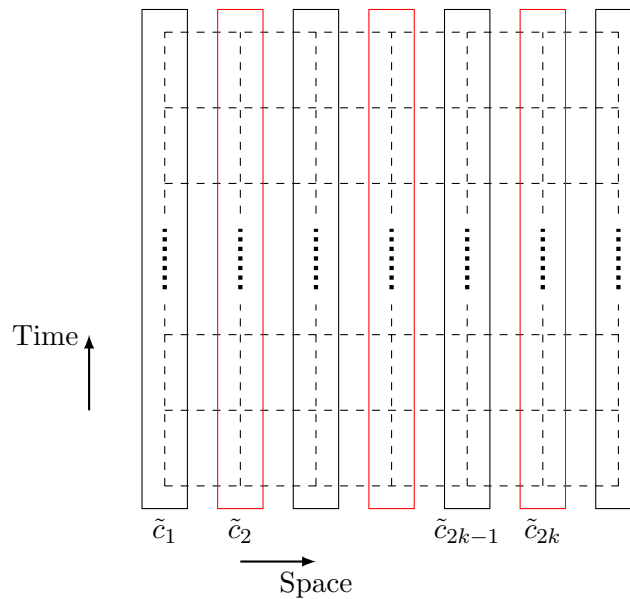


Figure 1.5: Schematic overview of the construction blocks in the Gauss Seidel line smoother

We repeated the same test as before where the preconditioner  $P$  is now defined by the above description.

Figure 1.6: Iterates after  $k = 0, 3, 8, 20$  iterations for the described line smoother, using a grid with  $11 \times 11$  elements. It needed more than about 1759 iterations in order for the norm of the residual to be  $< 10^{-9}$ . where  $A, f$  as before

As one might expect the line smoother converges much faster than the block Jacobi iteration, since the individual blocks we solve for exactly are larger. We can also see that the regions become more homogenous more quickly. *there are quite large areas where we go below 0, why?* But obviously this comes with an increase in the computational cost. Performs much better than 3 by 4, why? Gauss Seidel?

test case	no of fine elem	iter until conv	final res norm	avg conv rate
1	120	8	$6.25 \cdot 10^{-10}$	0.065
2	2	7	78	5415
3	0	0	0	0

Figure 1.7: test case 1:  $\mathcal{S} = (0, 1)$ , with  $u(0) = 1$ ,  $u(1) = 2$ , initial iterate  $u_{\text{init}} = 0.1$ ,  $\sigma_{\text{init}} = 0.1$ ,  $f = 0$ . test case 2:  $\mathcal{S} = (0, 3)$ , with  $u(0) = 3$ ,  $u(1) = 0$ , initial iterate  $u_{\text{init}} = 1$ ,  $\sigma_{\text{init}} = 0.5$ ,  $f = 0$ . test case 3: *put it if i get f nonzero working.*

### 1.3 Numerical Test Cases

#### 1.3.1 One-Dimensional Poisson Equation

After having discussed the individual multigrid terms let us look at the overall multigrid performance. As a very first test case we construct a one-dimensional Poisson problem using the least squares set up. Of course this problem could be solved more easily using for example a primal weak formulation however this can serve us as a simple test case for the multigrid implementation. We would like to remark here, that it necessarily has to be a one-dimensional problem in space, as we would otherwise have to use a different set of basis functions, like Raviart-Thomas basis functions, to obtain a well-defined discretisation, which would then not be using later on. Hence we want to find an approximate solution to the continuous minimisation problem

$$\begin{aligned} \text{Find } [\sigma, u] \in H_{\text{div}}(\mathcal{S}) \times H^1(\mathcal{S}) \text{ such that } \hat{J}([\sigma, u], f) \text{ is minimal, with} \\ \hat{J}([\sigma, u], f) = \frac{1}{2} \| -\text{div}(\sigma) - f \|_{L^2(\mathcal{S})}^2 + \frac{1}{2} \| \sigma - \nabla u \|_{L^2(\mathcal{S})}^2. \end{aligned} \quad (1.7)$$

where we assume  $f = f(x, t)$  and use the multigrid algorithm described above to solve the arising linear system of equations, only adapting the interpolation and restriction operators to suit a one-dimensional problem, standard one used here, see e.g. [?] for details.

The number of elements was chosen to roughly match the number of elements we will consider in the following sections to make the results more comparable, however as we only coarsen in space here. The iterations until convergence counts the number of V-cycles needed to read a residual norm smaller than  $10^{-9}$  using 3 pre-and post smoothing steps, using a 2 levels. The local convergence rate is defined as  $\mu_k = \frac{\text{res}_k}{\text{res}_{k-1}}$ , where  $k$  stands for the current iterate, and the overall convergence rate  $\mu$  is defined as its geometric mean over all iterations. Hence the smaller the value is, that is the closer to zero the better is the update. If we have  $\mu_k > 1$ , it means that the residual actually increased from one iteration to another. The first values of  $\mu_k$  highly depend on the initial guess but we can see below that later on that the average value is around 0.05 which is of course a very fast rate of convergence, however our test case is also very simple, so one would also expect the algorithm to converge very quickly.

Figure 1.8: Left-Panel: Convergence rate  $\mu_k$  for the first test case (see table),  $\mu =$ , Right-Panel: Convergence rate  $\mu_k$  for the second test case,  $\mu =$ .

The pictures of the overall solution are not provided here, but they match the expected solutions in all test cases. Hence we can conclude that for these simple test cases the set up of the least squares formulation as well as the multigrid implementation work *reasonably* well.

### 1.3.2 Heat Equation

Many parameters to test. Here we want to mainly focus on, grid size and grid size ratio, diffusion constant, and interplay  $c_1$  and  $c_2$  and then also how they relate to each other for a heat equation using the derived LSFEM space-time discretisation. Variety of parameters to be tuned. diffusion constant  
mesh size, ratios

play with  $c_1, c_2$ ?  
smoothers, more before?

We can see ....

Scaling tests for the multigrid, working with different grid sizes

Figure 1.9: residual norm over iterations. stopping criterion, residual norm smaller than  $10^{-9}$ ,  $\Omega = (0, 1)^2$ , same number of elements in time and space,  $D = 0.1$ ,  $c_2 = 10$ , 2 level methods, 3 pre -and post smoothing steps, block Jacobi,  $2 \times 2$ .

*Does not seem to scale very well ....?*

Also diffusion constant has a major effect on the speed of convergence. But relationship not that simple, diffusion constant does not occur in  $B_{\sigma\sigma}$ , otherwise it occurs in the term following terms  $-D\langle \nabla v, \tau \rangle$  in  $B_{\sigma u}$  and  $B_{u\sigma}$  and in  $B_{uu}$  as  $D^2\langle \nabla v, \nabla w \rangle$ . galerkin, not galerkin assembly

The choice of the parameters  $c_1$  and  $c_2$  has a major influence on the convergence speed of the multigrid algorithm. As we assume equivalence to the original partial differential equation the continuous optimisation problem  $J$  as well as  $\nabla J$  are equal to zero at the solution  $[\sigma, u]$ . However as we minimise the residual in a finite dimensional subspace and need to be taking rounding errors into account this is not *necessarily* true for the discretised problem. Therefore when we scale the problem

$$J([\sigma, u], 0) = \frac{1}{2}c_1 \|u_t - \text{div}(\sigma)\|_{L^2(\Omega)}^2 + \frac{1}{2}c_2 \|\sigma - D\nabla u\|_{L^2(\Omega)}^2. \quad (1.8)$$

on a finite mesh, we converge to a slightly different solution, depending on the parameter of  $c_2$ , *also because of the inexact convergence criterium? Why does it change solution so much? With increasing number of elements it will tend more and more towards correct solution?!*. Below we can see the behavior of the residual over iterations as well as the convergence rate for the same problem that only differs in the choice of  $c_2$ .

Figure 1.10: created from a fine mesh with 20 by 20 elements on a unit square,  $D = 0.1$ , block Jacobi smoother with block sizes of  $2 \times 2$ ,  $u_0 = \max(1 - 2x, 0)$  and Neumann boundary conditions in time

*What happens at the end there in case of convergence coefficient? Some trade off between convergence speed and accuracy? Sources? How is this solely related to eigenvalues and how much could it be other things?*

Complicated interplay between a number of parameters. If we change the diffusion constant  $D$ , it also massively effects the convergence speed.

GRAPHIC, residual norm  $D = 1$ ,  $D = 0.01$  for above  $c_2$  ...

sample	$N_x$	$N_t$	$D$	$c_2$	Iter until Conv	$\mu$	$\kappa(H)$
1	10	10	0.1	1	98	0.858	
2	20	20	0.1	1	205	0.900	
3	30	30	0.1	1	403	0.947	
4	10	10	1	1	91	0.790	
5	20	20	1	1	313	0.933	
6	30	30	1	1	641	0.966	
7	10	10	0.01	1	375	0.945	
8	20	20	0.01	1	$\approx 800$	$\approx 0.973$	
9	30	30	0.01	1	$> 800$	$> 0.981$	
10	10	10	0.1	10	36	0.551	
11	20	20	0.1	10	91	0.788	
12	30	30	0.1	10	213	0.902	
13	10	10	1	10	40	0.563	
14	20	20	1	10	116	0.822	
15	30	30	1	10	236	0.909	
16	10	10	0.01	10	274	0.925	
17	20	20	0.01	10	413	0.949	
18	30	30	0.01	10	740	0.971	
19	10	10	0.1	100	12	0.163	
20	20	20	0.1	100	25	0.418	
21	30	30	0.1	100	49	0.638	
22	10	10	1	100	169	0.862	
23	20	20	1	100	155	0.852	
24	30	30	1	100	179	0.871	
25	10	10	0.01	100	81	0.768	
26	20	20	0.01	100	84	0.772	
27	30	30	0.01	100	112	0.822	

Table 1.1: Overview of a number of sample cases, tested for varying parameters in the number of fine elements  $N_x$ , the number of fine elements in time  $N_t$ , the diffusion constant  $D$  and the scaling parameter  $c_2$ . The iterations until convergence give the number of V-cycles, using a 2-level method with 3 pre-and post smoothing steps of the block Jacobi smoother introduced before, with a patch size of  $2 \times 2$ , and  $\kappa(H)$  denotes the condition number of the symmetrised Hessian



then also test for different mesh sizes.

Table with varying parameters?

plot eigenvalues?

*parameters, boundary conditions, grid sizes, ...*

*pictures, this is how the solution looks like*

*some tables with convergence results* also how many steps do we need until convergence of the smoothers by itself

*further analysis? test laplace and  $u_t$  independently, what can we see from there?*

*Galerkin not Galerkin assembly, try with both* We have included the boundary values in the problem.

where is  $D$  and where is  $c_2$ , interplay .... ?!

Look at eigenvalues in comparison for those test cases that work well and those that don't really work at all. Also look at condition number of submatrices.

for the particular problem one is looking

**Eigenvalues + condition number** We looked at the eigenvalues of the hessian in order to better understand why the convergence is so slow. Below we can see the eigenvalues for the symmetrised hessian.

In comparison the eigenvalues of the one-dimensional Laplace equation describe a sine function [source]. We even have that the condition number here is higher.

Figure 1.11: Left-Panel: Sorted eigenvalues of the Hessian  $H$  using 11 elements in both space and time with  $\kappa(H) = 1.5 \cdot 10^3$ . Right-Panel: Sorted eigenvalues of the Hessian using 24 in space and 44 in time with  $\kappa(H) = 23 \cdot 10^3$ . In both cases  $\Omega = (0, 1)^2$ .

We can see that all eigenvalues are positive, as they have to be when arising from a least squares approach. And although it does not seem like the eigenvalues here can be described by a smooth function, they seem to be distributed fairly regularly and give rise to a *reasonable* condition number. *So that one would expect the multigrid to behave better What else does this tell us? What else can I say?*

### 1.3.3 Monodomain Equation

write that if we do simple Newton, non-convex iterations. makes sense to use trust region.

The simplified monodomain equation that shows the movement of an excitation front through space-time looks as follows

$$\begin{aligned} \partial_t u - \operatorname{div}(D \nabla u) &= u(1 - u)(\alpha - u), & D > 0, \quad 0 < \alpha < 1 \\ u(x, 0) &= u_0(x) & \forall x \in S \\ \nabla u \cdot n &= 0 & \forall (x, t) \in \{x \in \{0, S\}, t \in (0, T)\} \end{aligned} \quad (1.9)$$

We can see that the right-hand side is clearly nonlinear as it describes a third order polynomial. It can be shown that the equation has 3 fixed points, two stable fixed in zero and one, and an unstable fixed point in  $\alpha$  [?]. Hence depending on the initial conditions  $u_0$  the solution will tend to either zero or one, unless it is exactly  $\alpha$  and then through diffusion, very much dependent on diffusion constant though, if there is values larger than  $\alpha$  eventually lead to an activation of entire domain. Below we can see approximations of the solution.

Figure 1.12: For both graphs we have  $u_0 = \max(1 - 2x, 0)$ , homogeneous Neumann boundary conditions on the spatial boundaries of the domain, for 20 elements in space and 40 in time. On the left we have a diffusion constant  $D = 10^{-3}$ , whereas on the right we have  $D = 3 \cdot 10^{-3}$ .

We can see that this is only the excitation and not the depolarisation, (is maybe less of an interest as long as is it depolarises again eventually) *Picture earlier? What order?*

The two equations giving rise to the above solutions only differ in the choice of the diffusion constant  $D$ . As to be expected we can see that with a larger value of  $D$ , the activation happens much more quickly. We can also see that the front is not as steep, as we have a faster transport of electric potential through space.

As previously mentioned the solutions are obtained using a dogleg trust region method for the nonlinear iteration, which is a combination of a gradient descent and a Newton iteration. This is necessary as with a simple Newton method we have run into iterates that do not lead to a decrease but instead an increase of the functional, hence were tending towards a local maximum. *More specifically we have ... description of the algorithm. ONLY WORKS FOR LOCALLY POS DEF..... which we have. WHY NOT NEWTON UNLESS NOT SPD THEN GRADIENT DESCENT, maybe i can check for convexity in a point and if not just do gradient descent?!!*

We are using a quadratic model to approximate the functional  $J$ , that is we solve

$$\min_p m(p) = f + g^T p + \frac{1}{2} p^T H p \quad \text{s.t. } \|p\| \leq \Delta, p \in \text{span}\{g, H^{-1}g\}. \quad (1.10)$$

$$p^U = -\frac{g^T g}{g^T H g} g \quad (1.11)$$

$$\tilde{p}(\tau) = \begin{cases} \tau p^U & 0 \leq \tau \leq 1 \\ p^U + (\tau - 1)(p^B - p^U) & 1 \leq \tau \leq 2 \end{cases} \quad (1.12)$$

In the dogleg algorithm there are many parameters that need to be chosen, and could potentially be chosen to give better convergence results, here we have restricted ourselves to some that have shown themselves to perform well on previous problem [source] and not done an extensive amount of parameter tuning, as the overall aim was to develop a working nonlinear solver for now, but there are definitely many *opportunities* for further testing to improve the overall performance, which we will discuss in more details in the following chapter.

The discretisation of the least squares formulation uses the same approximation spaces and basis as introduced in the previous sections. The nonlinear part is computed as described in section 4.5 and 4.6. *MORE....?*

Below we can see two tables for the last iteration steps of the trust region method before reaching the convergence criterion. We can see that in a neighbourhood of the respective solutions, the quadratic model seems to be a good fit to the solution, as  $\rho$  is close to 1, which leads to a continually increasing trust region radius. We can also see that norm of the residual, that is  $\nabla J$  decreases at ... rate. *how to check that?!* The contains the number of accepted iterations, that is each iterations where an update took place, as well as the number of attempted iterations, this way one can get an insight into how often the model was determined to not be a good fit and the trust region radius was reduced without performing an update on the solution. *better to make graphs?!*

Accepted Iter	Overall Iter	$\ J\ $	$\ \nabla J\ $	$\rho$	TR Radius
111	152	$7.520771 \cdot 10^{-5}$	$2.3748 \cdot 10^{-4}$	0.835	192
112	153	$7.520542 \cdot 10^{-5}$	$2.3489 \cdot 10^{-6}$	1.0189	384
113	154	$7.520539 \cdot 10^{-5}$	$1.2572 \cdot 10^{-6}$	1.0053	768
114	155	$7.520539 \cdot 10^{-5}$	$7.1719 \cdot 10^{-11}$	0.9991	1536

Figure 1.13: Last steps of the Iteration, where  $\mathcal{S} = (0, 1)$ ,  $\mathcal{T} = (0, 20)$ , 20 elements in space, 40 elements in time, diffusion constant  $d = 1 \cdot 10^{-3}$ ,  $u_{\text{init}}(\cdot, t) = u_0$  for all time steps, and  $\sigma_{\text{init}} = 0$ , boundary conditions as before

accepted Iter	overall Iter	$\ J\ $	$\ \nabla J\ $	$\rho$	TR Radius
62	84	$1.50271 \cdot 10^{-4}$	$1.9825 \cdot 10^{-5}$	1.029	96
63	85	$1.50269 \cdot 10^{-4}$	$2.653 \cdot 10^{-5}$	1.0002	192
64	86	$1.50268 \cdot 10^{-4}$	$6.582 \cdot 10^{-9}$	1.00045	384
65	87	$1.50268 \cdot 10^{-4}$	$7.061 \cdot 10^{-12}$	1.14	768

Table 1.2: Last steps of the Iteration, where  $\mathcal{S} = (0, 1)$ ,  $\mathcal{T} = (0, 20)$ , 20 elements in space, 40 elements in time, diffusion constant  $d = 3 \cdot 10^{-3}$ ,  $u_{\text{init}}(\cdot, t) = u_0$  for all time steps, and  $\sigma_{\text{init}} = 0$ , boundary conditions as before

As before the two problems differ only in the diffusion constant. We can see that with a higher diffusion constant the problem converges faster. *Problem becomes more elliptic?!*  
How does mesh size affect this?

### 1.3.4 Linearisation of a Monodomain Equation

After considering the full nonlinear problem we particularly draw our attention to its linearisation in a neighbourhood of the solution  $s$ , the full description of how to obtain this solution we refer to the next section. Under the assumption that we have a unique local minimiser of the problem, we know that we have convexity close to the solution, that is the hessian  $H_s = H(s)$  is positive definite and the gradient is very close to zero.

In order to better understand the behavior of the hessian around the solution we perform an error analysis using a Jacobi smoother on the linearised system. More specifically we want to see how small perturbations in the solution in areas of the wavefront and the constant regions behave under the iteration of the smoother. *This could be in order to construct adapted coarse spaces that take these different local behaviors into account. talk more about why we have been looking at this.*

We consider the following iteration

$$\begin{aligned} e_{k+1} &= e_k + P(\nabla J - H_s e_k) \quad \text{or} \\ e_{k+1} &= e_k - P H_s e_k \end{aligned} \tag{1.13}$$

if we assume exactly that  $\nabla J = 0$ , and where  $P$  describes the preconditioner. We now let  $e_0$  equal to zero except for a small perturbation either in the area of the wavefront or the constant regions and see how this behaves under iteration.

On average the residual norm of the perturbed areas decrease by the following contraction factors

Figure 1.14: On the left, subsets of grid points on the constant zone, using solution for  $d = 1 \cdot 10^{-3}$ , and on the wavefront where we set the perturbations. On the right the residual norms after each iteration for one of the random test sets.

$$\|e_k^{const}\| \leq 0.075 \cdot 0.756^k, \quad \|e_k^{wf}\| \leq 0.068 \cdot 0.801^k, \quad 0 \leq k \leq 10 \quad (1.14)$$

*better to do a graph?!*

where  $k$  describes the number of iterations,  $e^{const}$  the perturbed vector on the constant area and  $e^{wf}$  on the wavefront, with random perturbations between 0 and 0.1 on the marked areas respectively. The numbers were computed averaging over a sample test set of size 100. We can see that perturbations on the wavefront are damped slower, which we can also see in figure 1.14, however this effect is not as drastic. Further experiments for differently sized patches and areas were conducted which lead to similar results.

$$\begin{aligned} u_{k+1} &= u_k - H_k^{-1}(\nabla J_k) \\ H_k u_{k+1} &= H_k u_k - \nabla J_k, \\ \text{let } r_k &:= H_k u_k - \nabla J_k \end{aligned} \quad (1.15)$$

As with most other numerical approximations there is a great number of parameters that could be tuned, choices that could have been made differently, which will be discussed in the following chapter.