Chapter 6

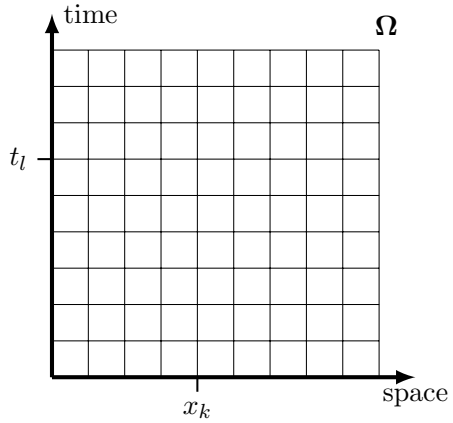
Implementation and Numerical Results

6.1 Discretisation Scheme

In this chapter we look at how the solution method we derived actually performs for on numerical test cases. We consider problems with a one-dimensional space domain $\mathcal{S} = (0, S)$ and a time interval $\mathcal{T} = (0, T)$, that is

$$\Omega = (0, S) \times (0, T), \quad S, T > 0, \tag{6.1}$$

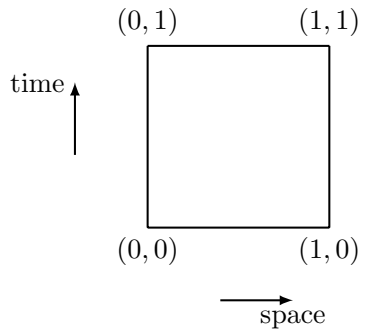
which we discretised using a uniform mesh with rectangular elements K_ζ, such that $\bar{\Omega} = \bigcup K_\zeta$.



which are defined by $N_x + 1$ nodes in space and $N_t + 1$ nodes in time. We will then refer to the tuples (x_k, t_l) in the following manner, the $\tilde{i}$-th entry references $\tilde{i} = (N_x + 1) \cdot l + k$. That is we first label all degrees of freedom belonging to a certain time step, such that $x_k < x_{k+1}$, before moving on to the next time step.

In our least-squares formulation we are dealing with the variables σ and u, which are both defined on Ω and for which we have to construct an overall approximation space. However we have seen before that they can be defined indepedently and then combined later on. We chose to use the same individual approximation space as well as basis functions for both of them. This is possible 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 beyond the scope of this thesis but would be of interest in future investigations. Therefore the following construction holds for each of them seperately and we will assemble the entire overall system later on.

As a finite-dimensional approximation space we used piecewise bilinear polynomials $Q_1(\Omega)$ in space–time. Given the reference element $\hat{K} = [0, 1]^2$ we used the following basis on $\hat{K}$



$$\begin{aligned}\hat{\phi}_{11}(x,t) &= (1-x) \cdot(1-t)\\ \hat{\phi}_{12}(x,t) &= x \cdot(1-t)\\ \hat{\phi}_{21}(x,t) &= (1-x) \cdot t\\ \hat{\phi}_{22}(x,t) &= x \cdot t\end{aligned} \tag{6.2}$$

add caption to $\hat{K}$: reference element $\hat{K}$.
Then the local space of the reference element is defined by $Q_1(\hat{K})$. There exist affine transformations $\{\hat{P}_\zeta : \hat{K} \to K_\zeta\}$ to each of the mesh cells K_ζ which look as follows

$$\hat{P}_\zeta \begin{pmatrix}\hat{x}\\ \hat{t}\end{pmatrix} = \begin{bmatrix}h_x & 0\\ 0 & h_t\end{bmatrix} \begin{pmatrix}\hat{x}\\ \hat{t}\end{pmatrix} + \begin{pmatrix}x_k\\ t_l\end{pmatrix}, \quad \text{for } (\hat{x},\hat{t}) \in \hat{K} \tag{6.3}$$

if K_ζ is determined by the coordinates $\{(x_k,t_l),(x_{k+1},t_k),(x_k,t_{l+1}),(x_{k+1},t_{l+1})\}$. And which define the local spaces

$$Q_1(K_\zeta) = \{\phi : \phi = \hat{\phi} \circ \hat{P}_\zeta, \hat{\phi} \in Q_1(\hat{\phi})\} \tag{6.4}$$

global basis

$$\phi_{ij}(x_k,t_l) = \begin{cases}1 & \text{if } (k,l) = (i,j)\\ 0 & \text{otherwise.}\end{cases} \qquad (k,l) \in \tag{6.5}$$

As previously mentioned σ and u are arranged 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 and u, respectively. The solution vector is therefore of size $2m$ while the matrices have a size of $2m \times 2m$. In order to compute the individual integral terms on the individual mesh cells we used Gaussian quadrature of order 3. The implemented admissible boundary conditions are a mixture of Dirichlet and Neumann type which get directly imposed in the system, respectively in either u or σ. Hence we now have all the necessary ingredients to compute the individual terms arising from the discretisation in Chapter 4.

6.2 Multigrid Implementation

We implemented a geometric multigrid V-cycle to solve the arising linear system of equations. It includes two different types of Vanka-smoothers which we will introduce below after discussing the coarse level construction. *what else can I say here?*

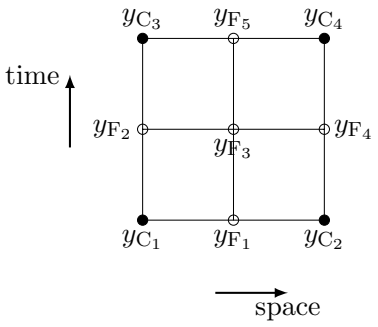
tried it with both Galerkin assembly and directly generating the operators on the coarse grid. lead to similar results.

6.2.1 Coarse Level Construction

An important question that always arises in every geometric multigrid implementation is how to choose the coarse level spaces. As discussed previously there is no unique optimal problem indepedent coarsening strategy, and especially in the case of space–time discretisations, the choice of the coarsening strategy has a great effect on the overall performance [25]. There is not yet much literature on what could be a favourable strategy in a least-squares space–time set up and therefore we decided to use a classical space–time coarsening approach, for each σ

and u independently. For each coarsening step the size of each rectangular element of in Ω grows by a factor of 4 with between each level while the length of the element 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 hence the degrees of freedom accordingly. The paper of [25] showed that space–time multigrid implementations for second order parabolic equations it is favourable to keep the quotient $\lambda = D\frac{\Delta t}{\Delta x^2}$ (where D decribes the diffusion constant and Δt and Δx the respective step sizes) close to one on all levels. However, as we are dealing with a mixed first order system here it seemed reasonable to not take this assumption into account but instead aim for $\lambda = D\frac{\Delta t}{\Delta x} \approx 1$, which is in line with the coarsening strategy described above. In section (6.3.2) we will see some numerical test cases to investigate this assumption. The implementation is then such that one chooses the number elements on the coarse grid and the number of levels. And subsequently all other meshes, interpolation and restrictions operators, and finer level operators are generated automatically, according to the chosen paramaters, which guarantees for nested meshes.

One still has to choose interpolation operators to project between levels, which we did according to the following scheme.



add caption. filled circles represent coarse grid points and the blank circles fine grid points. y stands for the values at each of the nodes.

$$
\begin{aligned}
s(y_{C_i}) &= s(y_{C_i}) \qquad \forall i \\
s(y_{F_1}) &= \frac{1}{2}y_{C_1} + \frac{1}{2}y_{C_2} \\
s(y_{F_2}) &= \frac{1}{2}y_{C_1} + \frac{1}{2}y_{C_3} \\
s(y_{F_3}) &= \frac{1}{4}y_{C_1} + \frac{1}{4}y_{C_2} + \frac{1}{4}y_{C_3} + \frac{1}{4}y_{C_4} \\
s(y_{F_4}) &= \frac{1}{2}y_{C_2} + \frac{1}{2}y_{C_4} \\
s(y_{F_5}) &= \frac{1}{2}y_{C_3} + \frac{1}{2}y_{C_4}
\end{aligned}
\tag{6.6}
$$

If $\tilde{I}$ is an interpolation operator of the above type is defined between two levels then they can be applied for either σ or u. Hence in order to obtain an interpolation for both we assemble them in the following form

$$
I = \begin{bmatrix} \tilde{I} & 0 \\ 0 & \tilde{I} \end{bmatrix} \tag{6.7}
$$

That is the two variables are interpolated indepedently, which seems reasonable, since on the one hand they are clearly related but as one is the spatial derivative of the other, this is difficult to express in an interpolation operator. The arising 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. Below we can see example of the application of an interpolation operator above type $\tilde{I}$.

Remark: In the implemenation of the multigrid V–cycle we remove the influence of the boundary conditions as in linear multigrid we apply the coarse grid correction on the error and therefore cannot have a correction coming from a fixed boundary.

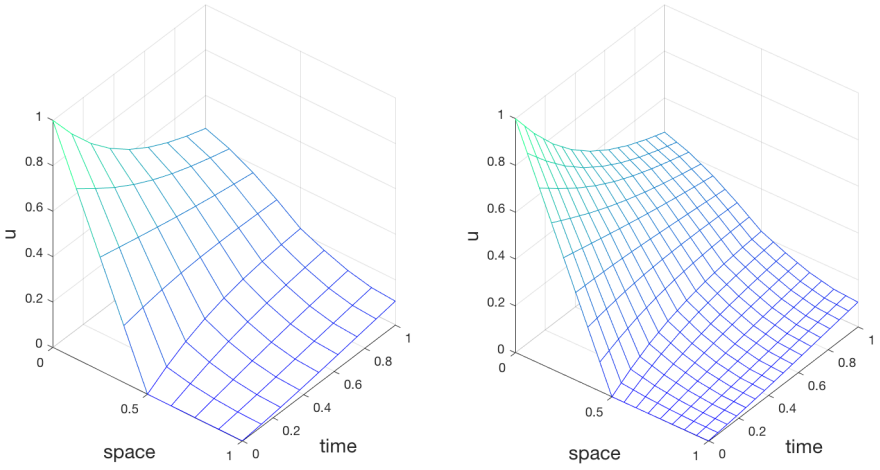


Figure 6.1: Interpolation of the solution to the heat equation with $u(x,0) = \max(1-2x,0)$, $D(x) = 0.1$ and homogeneous Neumann boundary conditions in time on a unit square from 8×8 to 16×16 elements. The interpolated solution of σ is not shown here as $\tilde{I}$ simply applied to σ and u independently.

6.2.2 Smoothers

Smoothers are a key part of a multigrid algorithm. We would like that the combination of coarse grid correction and smoother reduces the error efficiently. But as this is a complicated interplay of two processes it is often not easy to determine in advance what would be a suitable choice for a smoother. 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 either of the two combined with the coarse grid correction are a favourable synthesis. Since we are also trying to develop a parallelisable solver it is of course important to also be taking that into account when choosing a smoother. Thus 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 σ_h and u_h, which are separated in the matrix-vector assembly as 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?] damped block Jacobi relaxation. That is a damped block Jacobi smoother where each block contains all degrees of freedom associated to one grid point in the space–time domain. One can choose p degrees of freedom in space and q in time, to obtain a rectangular patch with $p \times q$ degrees of freedom on the domain. Then the algorithm constructs a submatrix containing all entries of A associated to those nodes for σ and u as well as all of the corresponding coupling terms and directly inverts the small submatrix. The entire domain is partitioned in this way. On the boundary we might potentially obtain smaller patches, depending on the divisibility of the chosen patch size. In this way we obtain a preconditioner P containing the inverted submatrices. Due to the organisation of σ and u, 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 assosciated with the patch, and C_i the corresponding submatrix, which are each of size 8×8.
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

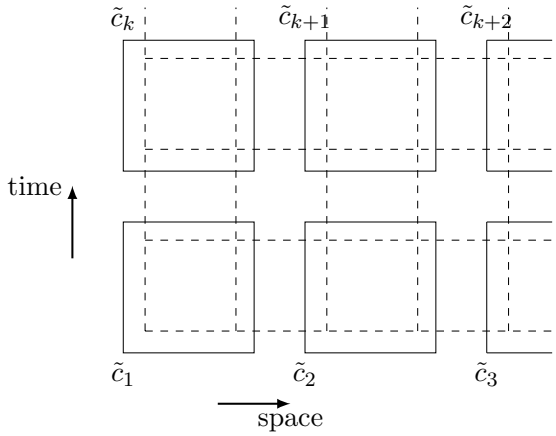


Figure 6.2: Schematic overview of the construction of the blocks of damped Jacobi smoother

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

$$e_{k+1} = e_k + \omega P(f - Ae_k) \tag{6.8}$$

where P is the preconditioner and ω 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.
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. We again only plotted the results for u as σ behaves in the same way, simply with different boundary conditions. We also compared the performance of the this smoother for different block sizes. As one would expect it reduces the error faster for larger block sizes. We can see that by comparing the number of overall iterations until the stopping criterion was met as well as by looking at the graphs which show less oscillations for the larger patch as well as values closer to zero. The damping factor was $\omega = \frac{2}{3}$ in both cases.

Another type of smoother that we tested is a Gauss-Seidel line smoother which is similar to works of [33] and can also be categorised as a Vanka smoother. One considers each spatial degree of freedom individually but together for all times as well as the values for σ 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.
We repeated the same test as before where the preconditioner P is now defined by the above description.
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?

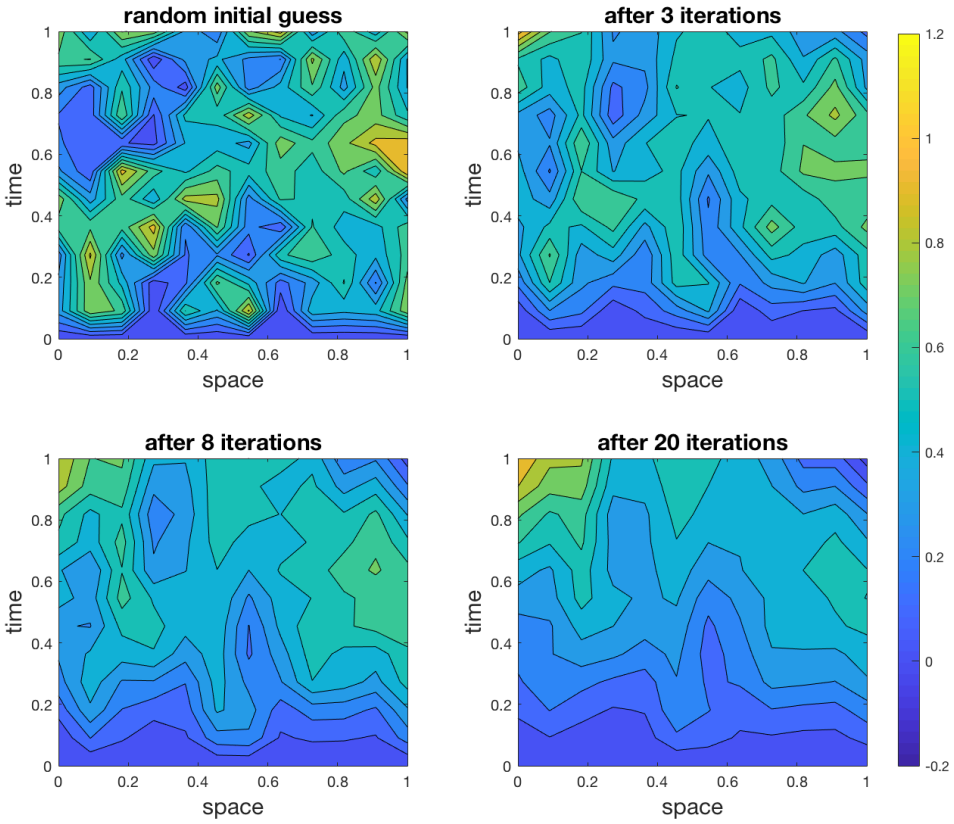


Figure 6.3: Iterates after $k = 0, 3, 8, 20$ iterations for a patch size of 1×1, and a grid with 11×11 elements. needed more than 9000 iterations in order for the norm of the residual to be smaller than 10^{-9}.

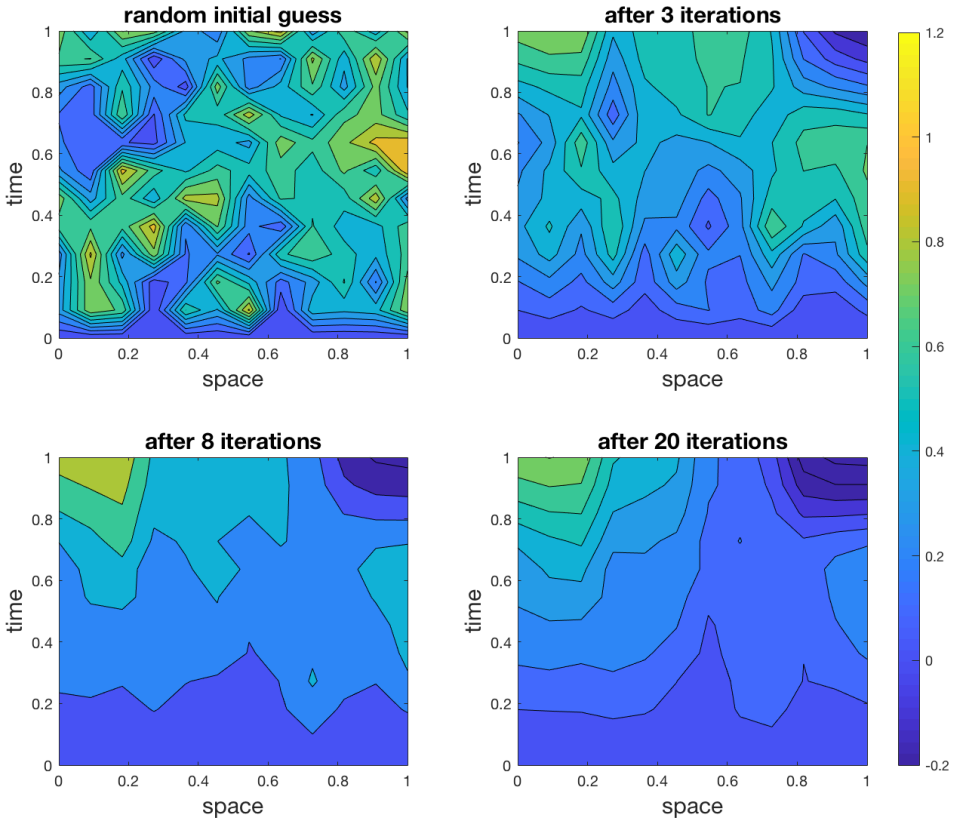


Figure 6.4: Iterates after $k = 0, 3, 8, 20$ iterations for a patch size of 3×4, and a grid with 11×11 elements. needed more than 3177 iterations in order for the norm of the residual to be smaller than 10^{-9}.

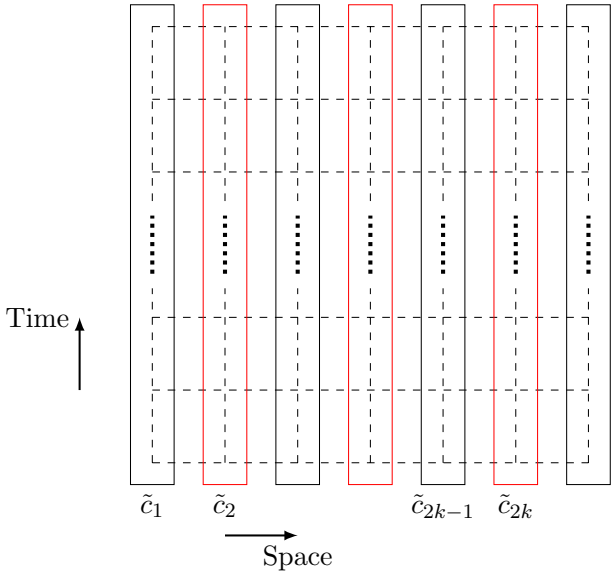


Figure 6.5: Schematic overview of the construction blocks in the Gauss Seidel line smoother

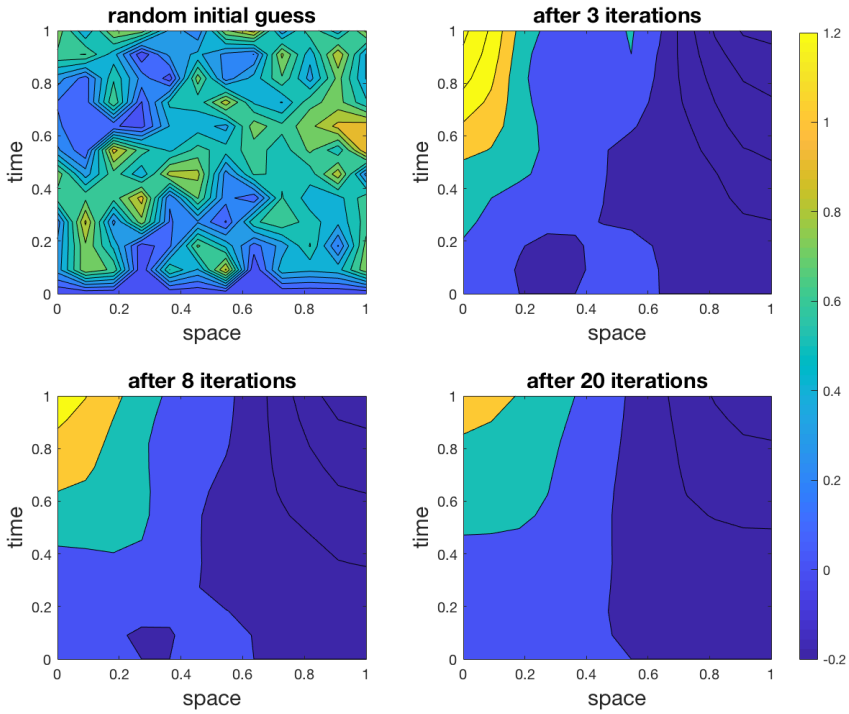


Figure 6.6: Iterates after $k = 0, 3, 8, 20$ iterations for the described line smoother, using a grid with 11×11 elements. It needed more than about 1759 iterations in order for the norm of the residual to be $< 10^{-9}$. where A, f as before

6.3 Numerical Test Cases

6.3.1 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 an 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

$$\begin{aligned} &\text{Find } [\sigma, u] \in H_{\text{div}}(\mathcal{S}) \times H^1(\mathcal{S}) \text{ such that } \hat{J}([\sigma, u], f) \text{ is minimal, with} \\ &\hat{J}([\sigma, u], f) = \frac{1}{2}|| - div(\sigma) - f||^2_{L^2(\mathcal{S})} + \frac{1}{2}||\sigma - \nabla u||^2_{L^2(\mathcal{S})}. \end{aligned} \tag{6.9}$$

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. [27] 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 $\mu_k = \frac{res_k}{res_{k-1}}$, where k stands for the current iterate, and the overall convergence rate μ 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 μ_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.
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.

test case	degrees of freedom	iter until conv	avg conv rate
1	200	9	$3.81 \cdot 10^{-2}$
2	500	9	$3.82 \cdot 10^{-2}$
3	2000	10	$4.59 \cdot 10^{-2}$
4	5000	10	$5.03 \cdot 10^{-2}$

Table 6.1: tall test cases on $\mathcal{S} = (0, 3)$, with $u(0) = 3$, $u(1) = 0$, initial iterate $u_{\text{init}} = 1$, $\sigma_{\text{init}} = 0.5$, $f = 0$ and only vary in the number of degrees of freedom, that is the resolution of the mesh. Smoother: Jacobi-smoother, patch size 1×1. Convergence criterium : norm of residual smaller than 10^{-9}.

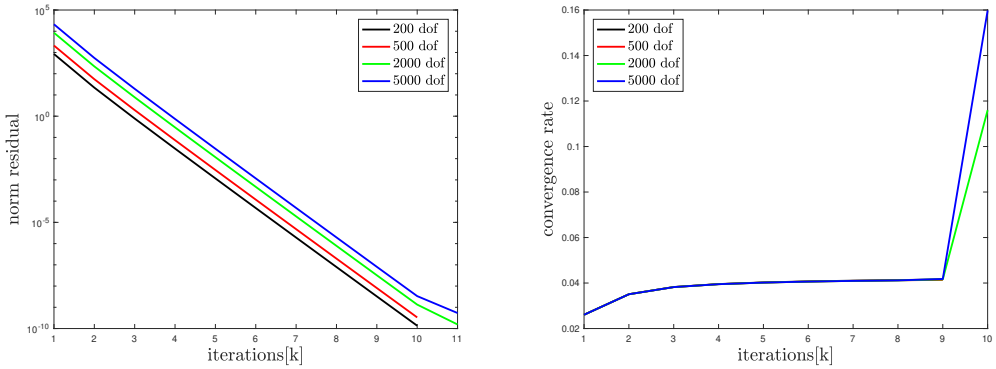


Figure 6.7: Left-Panel: norm of the residual over iterations for test cases 1 to 4, Right-Panel: Convergence rate μ for test cases 1 to 4. See Table (6.1) for details.

6.3.2 Heat Equation

In this section we tested the multigrid implementation on our least-squares space–time finite element formulation imposing a zero forcing term. The continuous form of the functional which we want to minimise is shown here again

$$J([\sigma, u], 0) = \frac{1}{2}c_1||u_t - div(\sigma)||^2_{L^2(\Omega)} + \frac{1}{2}c_2||\sigma - D\nabla u||^2_{L^2(\Omega)}. \tag{6.10}$$

The multigrid algorithm converged for all tested input parameters. However there seem to be a lot different factors influencing the speed of convergence. The figure below shows the norm of the residual over the number of iterations for different numbers of degrees of freedom.

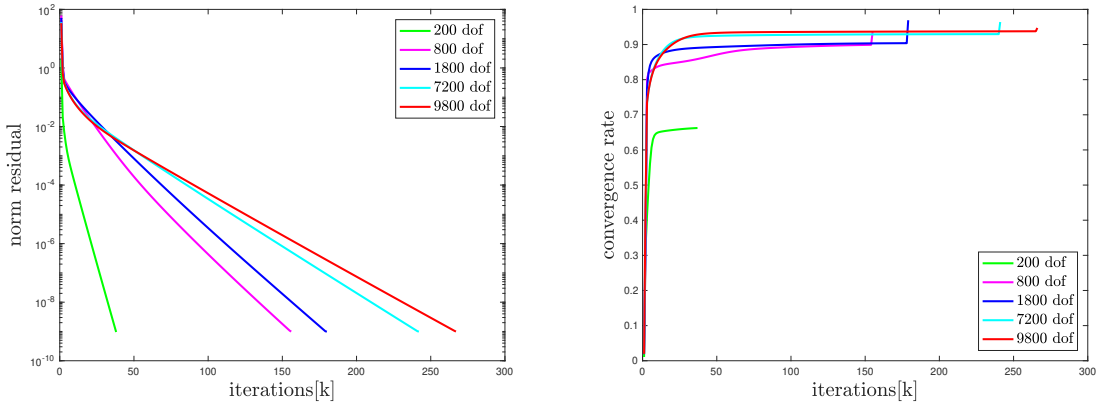


Figure 6.8: Left-Panel: Norm of the residual over iterations. Right-Panel: Convergence rate μ. Stopping criterion residual norm smaller than 10^{-9}, $\Omega = (0,1)^2$, same number of elements in time and space, $d = 1$, $c_2 = 100$, 2 level methods, 3 pre –and post smoothing steps, block Jacobi, patch size 2×2.

In general it also seems that the required number of iterations until we reach the stopping criterion is quite large even though the tested problems are still very small. *Problem sort of scales but not great. more later.* In order to find out what might be the cause, we varied different input parameters to see how it would affect the convergence results. The ones we want

to mainly focus on are grid size and grid size ratio, a varying diffusion constant, and the interplay of the scaling factors of c_1 and c_2. We can find a table below that lists different parameters properties as well as the corresponding convergence results, like the number of required V-cycle and the convergence rate. This is meant to give an overview of the behavior of this solution method. Subsequently we will go into more detail concerning the affects of the individual input parameters to gain a better understanding for each of them.
We can see that number of iterations needed until the convergence criterion is met differs greatly. Changes in some paramters seem to have a straight forward correlation with the convergence speed while with others this does not appear to be the case. So let us discuss them in more detail one by one.

We begin by considering the changes in the parameter c_2. As we are only interested in the different ratios of c_1 and c_2 it suffices to change one of the parameters while leaving the other one constant. We can see in Table (6.2) that a larger coefficient c_2 seems to reduce the convergence rate, while leaving all other parameters the same. And this behavior is indepedent of all other parameters that we tested. We can also see in the last column that a larger coefficient c_2 leads to a smaller condition number of the matrix H. Smaller condition numbers often seem to lead to better convergence rates except for a few cases which we will discuss in more detail later. And where we will check how the above terms influence the condition number of H and how this is potentially related to the convergence speed.

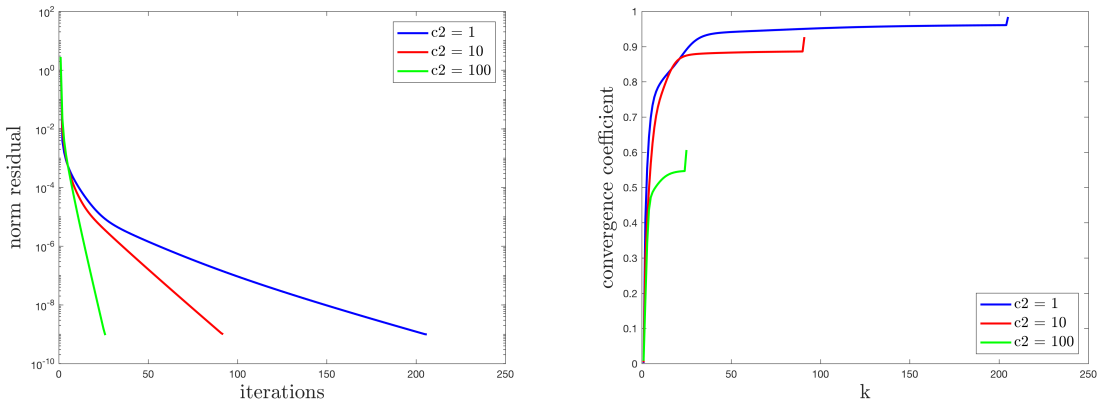


Figure 6.9: created from a fine mesh with 20 by 20 elements on a unit square, $D = 0.1$, block Jacobi smoother with block sizes of 2×2, $u_0 = max(1-2x, 0)$ and Neumann boundary conditions in time. The increase in the end may be caused by cancellation of leading digits as the norm of the residuals for the final iterates become very small.

Question: 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? Also see if the smoother reacts to scaling?
The scaling parameters in the continuous problem formulation (6.10) do not affect the overall minimiser as we assume J to be zero at the solution $[\sigma, u]$. However this is not necessarily the case in the discrete setting. The discretised problem does not necessarily have exactly zero as a minimiser. The minimiser for each of the two discrete terms may be slightly different. By changing the coefficient c_2 we put more emphasis on being closer to the discrete minimiser of the second term than on the first. It is however not obvious how this relates to finding the discrete solution that resembles the continuous one best. Nevertheless it slightly changes the final solution we obtain. For any fixed ratio of c_1 and c_2 we will nevertheless converge to the

sample	N_x	N_t	d	c_2	Iter until Conv	μ	$\kappa(H)$
1	10	10	0.1	1	98	0.858	1251
2	20	20	0.1	1	205	0.900	4707
3	30	30	0.1	1	403	0.947	10296
4	10	10	1	1	91	0.790	312
5	20	20	1	1	313	0.933	1505
6	30	30	1	1	641	0.966	3675
7	10	10	0.01	1	375	0.945	3561
8	20	20	0.01	1	≈ 800	≈ 0.973	19735
9	30	30	0.01	1	> 800	> 0.981	55902
10	10	10	0.1	10	36	0.551	214
11	20	20	0.1	10	91	0.788	765
12	30	30	0.1	10	213	0.902	1651
13	10	10	1	10	40	0.563	1902
14	20	20	1	10	116	0.822	7090
15	30	30	1	10	236	0.909	15521
16	10	10	0.01	10	274	0.925	1084
17	20	20	0.01	10	413	0.949	4115
18	30	30	0.01	10	740	0.971	8977
19	10	10	0.1	100	12	0.163	215
20	20	20	0.1	100	25	0.418	732
21	30	30	0.1	100	49	0.638	1574
22	10	10	1	100	169	0.862	19006
23	20	20	1	100	155	0.852	70890
24	30	30	1	100	179	0.871	155190
25	10	10	0.01	100	81	0.768	322
26	20	20	0.01	100	84	0.772	895
27	30	30	0.01	100	112	0.822	1792

Table 6.2: 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 2×2, and $\kappa(H)$ denotes the condition number of the symmetrised Hessian

test case	d	c_2	$\kappa(H)$	$\kappa(H_{\sigma\sigma})$	$\kappa(H_{uu})$	smallest eigenvalue	Iter until Conv
2	0.1	1	4707	584	1786	$8.4 \cdot 10^{-4}$	205
5	1	1	1505	584	1416	$2.6 \cdot 10^{-3}$	313
11	0.1	10	764	320	712	$5.2 \cdot 10^{-3}$	91
20	0.1	100	731	59	711	$5.6 \cdot 10^{-3}$	25

Table 6.3: More detailed table with condition numbers. d describes the diffusion constant. c_2 scaling parameter of the second term of the functional. $\kappa(H)$: condition number of the Hessian. $\kappa(H_{\sigma\sigma})$: condition number of the submatrix $H_{\sigma\sigma}$, and for H_{uu} respectively. All test cases are of the same size.

continuous solution by increasing the resolution of the mesh.

$$H_{\sigma\sigma} : c_2\langle\phi_i, \phi_j\rangle, \quad H_{\sigma u} : c_2\langle(\phi_i)_t, (\phi_j)_x\rangle, \quad H_{u\sigma} : c_2\langle(\phi_j)_t, (\phi_i)_x\rangle \quad H_{uu} : c_2\langle(\phi_i)_x, (\phi_j)_x\rangle \tag{6.11}$$

The diffusion constant also has a major effect on the speed of convergence. However in contrast to the paramter c_2 it also changes the overall continuous solution. And the influence it was is not that straight forward to see. Therefore we will now focus our attention on an analysis of the eigenvalues and the condition number of the arising matrices.
In order to do so we look at the eigenvalues in comparison for those test cases that work well and those that don't really work at all.
We also looked at the eigenvalues and condition numbers of the Hessians for different parameters in order to better understand if this could potentially be related to the slow rate of convergence. It makes sense to consider them together as for regular symmetric matrices we have that $\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)}$. The table below contains more information on the condition numbers of the Hessians and its submatrices. The numbers in the test case column correspond to those test cases from Table (6.2).
We can see from Table (6.3) that there seems to be correlation between the condition number and the iterations until convergence for some cases but not for all. When we look at the condition number of the individual blocks, then $H_{\sigma\sigma}$ usually has a much smaller condition number compared to H_{uu}. The off-diagonal blocks are singular and therefore we cannot consider them by themselves. If we now also take the graphs below under consideration, we can see that the eigenvalues mostly vary in their small eigenvalues. But this cannot be the only factor as we can see from the fact that test case two took more iterations until convergence than test case five, even though it has a much larger condition number. Additionally we can see that test case 11 and 20 almost have the same overall condition number, but the condition number of the $H_{\sigma\sigma}$ blocks differ a lot, as well as the required iterations until convergence.
In Figure (6.11) we can see the eigenvalues of the Hessians of the same test cases as well as the eigenvalues of the iteration matrix of the relaxation scheme. We can see that the larger eigenvalues of the Hessian seem to be pretty identical in all four cases. Additionally we can say that the distribution of the eigenvalues can probably not be described by a smooth function, as in the case of the Laplacian however, they seem to be distributed fairly regularly. The plateau at one is due to the enforced boundary conditions. All of the Hessians give rise to *reasonable* condition numbers, as we have seen in the table. Now we also considered the eigenvalues of the iteration matrix of the relaxation scheme using the 2×2 Jacobi block-smoother, which can be seen on the left. Due to the damping factor $\omega = 0.64$ the plateau of the imposed boundary conditions is not at zero as one would expect. If we then consider test cases 22 to 24 in comparison things look entirely different. Here the condition number seems to hardly affect the number of iterations.

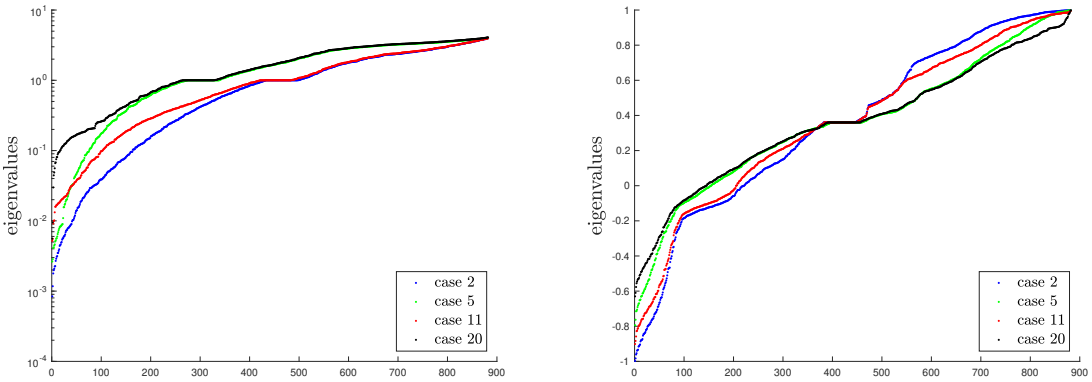


Figure 6.10: Left-Panel: Sorted eigenvalues of the Hessian of the test cases $2, 5, 11$ and 20, using a logscale on the y-axis. Right-Panel: Sorted eigenvalues of the iteration matrix $M = (I - PH)$ for the same test cases, where P is the preconditioner arising from the block Jacobi smoother using patches of size 2×2.

Figure 6.11: Left-Panel: Sorted eigenvalues of the Hessian of the test cases $22, 23$ and 24, using a logscale on the y-axis. Right-Panel: Sorted eigenvalues of the iteration matrix $M = (I - PH)$ for the same test cases, where P is the preconditioner arising from the block Jacobi smoother using patches of size 2×2.

We also tested the algorithm using different initial iterates for σ and u. In the cases shown above we initially set $\sigma(x_k, t_l) = 0$ for all (k, l) and $u(\cdot, t_l) = u_0$ for all l. However this did not seem to have a big effect on the convergence speed as we also tested for other initial iterates which all lead to very similar or the same number of iterations until convergence. This is in line with the above results in the sense that the first iterations the residual reduction happens quite quickly and then slows down a lot.

We additionally compared the usage of either a coarse grid operator constructed by a Galerkin projection through the interpolation and restriction as well as a direct assembly of the Hessian on the coarse grid. They performed similarly although the Galerkin assembly tends to lead to a slighlty better convergence rate. This is probably due to the fact that it represents the actual projection of the fine grid problem and not a separate related problem.

From the analysis done so far it is yet not very clear where exactly the slow rate of convergence stems from. We have seen in Figure (6.8) that the multigrid algorithm actually reduces the norm of the residual in every iteration. The interpolation operator we apply also seems to be working correctly, see Figure (6.1). We have also tested the behavior of the smoother as well as the distribution of their eigenvalues. The smoother is displaying a notable smoothing property, see Figures (6.3) and (6.4). The eigenvalues of the arising iteration matrix seems *normally* distributed. However it seems that depending on certain parameters some error modes are not reduced efficiently. We can remark that a larger parameter c_2 improved the performance for all test cases. The dependency on the other parameters we tested is not as straight forward. We will be discussing possibilities what this might be related to and how to overcome these problems in Chapter 7.

6.3.3 Monodomain Equation

After having looked at some linear test cases for the multigrid algorithm we will now examine the behavior of a nonlinear space–time least–squares finite element discretisation for the test case of a simplified monodomain equation. A simplified monodomain equation that shows the movement of an excitation front through space–time looks as follows

$$\begin{aligned} \partial_t u - div(d\nabla u) &= u(1-u)(u-u_{\text{thres}}), && \text{in } (0,S)\times(0,T), \\ u(x,0) &= u_0(x) && \text{on } (0,S), \\ \nabla u \cdot n &= 0 && \text{on } (0,T) \text{ for } x \in \{0,S\}, \end{aligned} \tag{6.12}$$

for $d > 0$, and $0 < u_{\text{thres}} < 1$. 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 u_{thres} [16]. Hence depending on the initial conditions u_0 the solution will tend to either zero or one, unless it is exactly equal to the value of the unstable fixed point u_{thres}. If there is values larger than u_{thres} that tend to one, we can observe that through the diffusive term this leads to an activation of the surrounding areas in space over time. The value of the diffusion constant d governs the *speed* diffusive reaction. This behavior can also be seen below in the graphs showing approximations of the solution. As mentioned before we can see that the above equation (6.12) only captures the excitation phase and not the depolarisation. The excitation of the tissue is the crucial part of the process and neglecting the depolaristion simplifies the model. The two equations giving rise to the

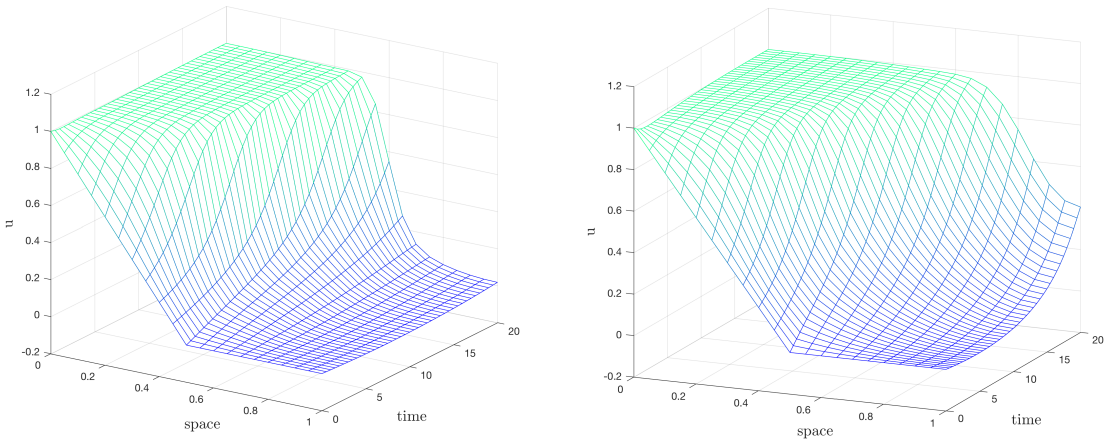


Figure 6.12: For both graphs we have $u_0 = \max(1-2x, 0)$, homogeneous Neumann boundary conditions on the spatial boundaries of the domain, for 20 elements in space and 40 in time. Left-Panel: diffusion constant $d = 10^{-3}$. Right-Panel: $d = 3 \cdot 10^{-3}$.

transmembrane potentials in Figure (6.12) 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 the electric potential spreads much faster in space.

As previously mentioned the solutions are obtained using a dogleg trust region method, which is a combination of a gradient descent and a Newton iteration. When only using a Newton method there are iteration steps where we run into locally non-convex areas and therefore take iteration steps towards local maxima. Through the usage of a trust region method, where the algorithm

first checks efficacy of the potential next step we have not experienced the same problem, but instead for each iterate obtained local convexity.
In the dogleg algorithm there is a variety of threshold 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] (*list of parameters will follow*) 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.
Below we can see two tables for the last iteration steps of the trust region method before reaching the convergence criterion. The parameters correspond to those depicted above in Figure (6.12). We can see that in a neighbourhood of the respective solutions, the quadratic model seems to be a good fit to the solution, as ρ is close to 1, which leads to a continually increasing trust region radius. We can also see that norm of the residual, that is ∇J decreases quickly, as one would expect when approaching the minimum of a convex neighbourhood. The tables also contain the number of accepted iterations, that is each iterations where an update actually 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.

Accepted Iter	Overall Iter	$\|\|J\|\|$	$\|\|\nabla J\|\|$	ρ	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

Table 6.4: 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	$\|\|J\|\|$	$\|\|\nabla J\|\|$	ρ	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 6.5: 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. In order to see how the diffusion term affects the complexity of the problem, tested it for more parameters.

accepted Iter	overall Iter	$\|J\|$	$\|\|\nabla J\|\|$	ρ	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 6.6: Testing the effect of the diffusion constant d on the number of iterations until stopping criterion is met. $\mathcal{S} = (0, 1)$, $\mathcal{T} = (0, 20)$, 20 elements in space, 40 elements in time, $u_{\text{init}}(\cdot, t) = u_0$ for all time steps, and $\sigma_{\text{init}} = 0$, boundary conditions as before.

We saw in Figure (6.12) that with an increasing diffusion constant the overall solution has less steep gradients. more One could suppose that *Problem becomes more elliptic?!*.

We also tested how the resolution affected the number of energy minimisation steps needed to reach the stopping criterion.

space int	N_x	time int	N_t	accepted Iter	overall Iter	$\|\nabla J\|$
$(0, 1)$	20	$(0, 10)$	40	41	74	
$(0, 1)$	20	$(0, 20)$	40	65	87	
$(0, 1)$	20	$(0, 20)$	80	84	148	
$(0, 1)$	40	$(0, 20)$	80	57	92	
$(0, 2)$	40	$(0, 20)$	80	73	101	

Table 6.7: different resolution and different domain sizes, diffusion constant $d = 3 \cdot 10^{-3}$, $u_{\text{init}}(\cdot, t) = u_0$ for all time steps, and $\sigma_{\text{init}} = 0$. With the same boundary conditions as before. Stopping criterion $\|\nabla J\| < 10^{-9}$

.
We can conlcude that the arising energy minimisation problem from the discretised monodomain equation has a rather complicated non-convex solution landscape. As we increase the impact of the diffusive term it converges faster which one could suppose implies that the problem becomes "more elliptic"? *Can I say that? And what else should I say here?*

6.3.4 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. Our aim is for one to better understand the properties governing the behavior of the solution. And secondly to have a closer look at how the multigrid algorithm performs in this case, to see if we behavior we have seen in the previous section on the heat equation repeats itself here. Or if due to the additional linearised forcing term we observe different dynamics. 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. From the previous section we know that this really is the case, see Table (6.4) and Table (6.5).
picture, residual convergence MULTIGRID to follow

Additionally 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 pertubations in the solution in areas of the wavefront and the constant regions behave under the iteration of the smoother. One of the intentions behind this idea was to potentially construct adapted coarse level spaces that take these different local behaviors into account. We expect the pertubations on the wavefront to be damped much slower than on the areas that have converged to the two stable fixed points. We therfore consider the following iteration

$$\begin{aligned} e_{k+1} &= e_k + P(\nabla J - H_s e_k) \quad \text{or} \\ e_{k+1} &= e_k - P H_s e_k \end{aligned} \tag{6.13}$$

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 pertubation either in the area of the wavefront or the constant regions and see how this behaves under iteration.

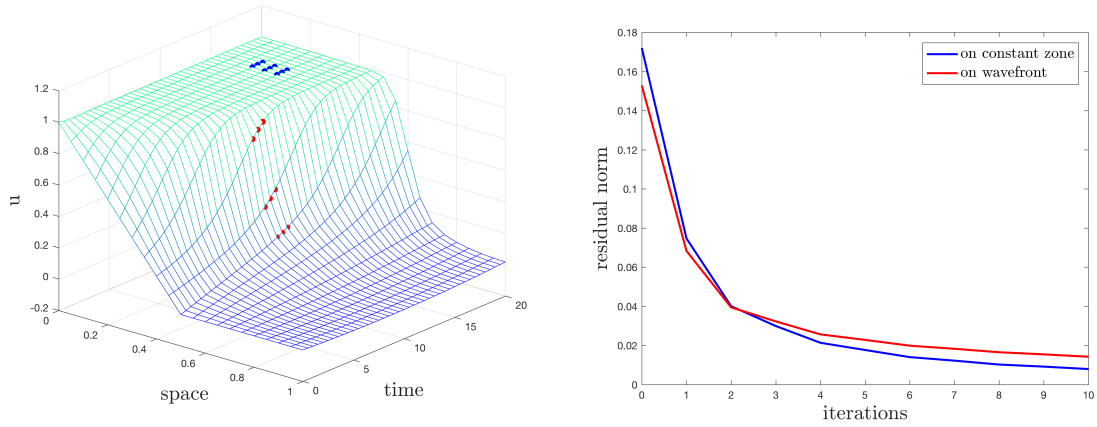


Figure 6.13: On the left, subsets of grid points on the constant zone, using solution for $d = 1 \cdot 10^{-3}$, and on the wavefront where we set the pertubations. On the right the residual norms after each iteration for one of the random test sets.

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

$$||e_k^{const}|| \leq 0.075 \cdot 0.756^k, \qquad ||e_k^{wf}|| \leq 0.068 \cdot 0.801^k, \quad 0 \leq k \leq 10 \tag{6.14}$$

where k describes the number of iterations, e^{const} the perturbed vector on the constant area and e^{wf} on the wavefront, with random pertubations 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 pertubations on the wavefront are damped slower, which we can also see in figure 6.13, however this effect is not as drastic. Further experiments for differently sized patches and on different areas of the domain were conducted which lead to similar results.
So what to take from this, how to write this up better?