Picard-Kachanov type iterations for nonlinear elliptic PDEs

Student: András Sike **Supervisor:** János Karátson

1 Introduction

Last semester we were taking a look at the nonlinear elliptic partial differential equation for the function $u : \Omega \to \mathbb{R}$

$$\begin{cases} -\mathrm{div}(A(x,u) \cdot \nabla u) + q(x,u)u = f; \quad \text{in } \Omega \\ u\big|_{\Gamma_D} = 0; \\ (A(x,u)\nabla u \cdot \nu + pu)\big|_{\Gamma_N} = g; \end{cases}$$

and investigated the Picard-Kachanov iteration (cf. [8],[10] for a theoretical discussion of the method), that is, we have reduced the problem to the following series of linear equations:

$$\begin{cases} -\mathrm{div}(A(x,u^{(n)}) \cdot \nabla u^{(n+1)}) + q(x,u^{(n)})u^{(n+1)} = f; \quad \text{in } \Omega \\ u^{(n+1)}\big|_{\Gamma_D} = 0; \\ (A(x,u^{(n)})\nabla u^{(n+1)} \cdot \nu + pu^{(n+1)})\big|_{\Gamma_N} = g; \end{cases}$$

where some 'previous' solution $u^{(n)}$ is given, and we are solving for $u^{(n+1)} : \Omega \to \mathbb{R}$. With the appropriate conditions, this problem becomes well-posed and we can solve the above equation by any method we wish, we have chosen the finite element method.

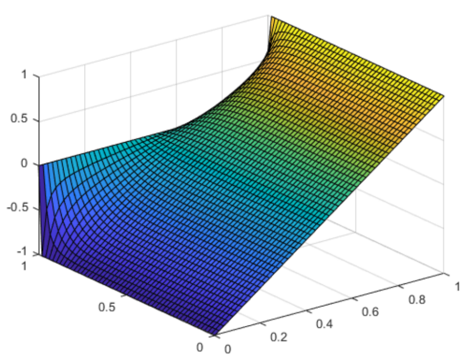
Before our investigations with FEM, we would also like to mention that we have also implemented the streamline-diffusion finite element method, that helps us solve a different type of problem, which we will not be focusing on:

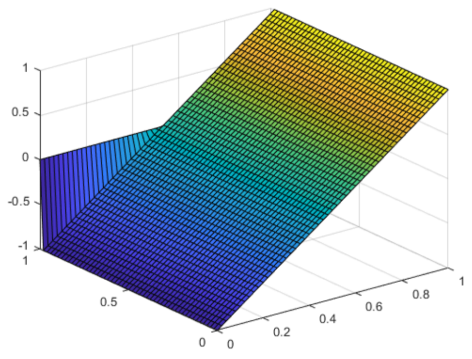
$$-\varepsilon \cdot \mathrm{div}(A(x,u) \cdot \nabla u) + w \cdot \nabla u = f,$$

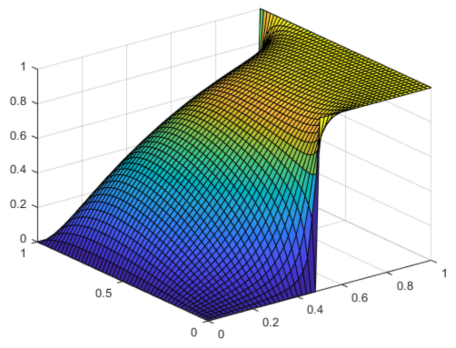
the so-called *convection-diffusion equation*, which is also well-posed with appropriate boundary conditions, as well as conditions on the vector field w. The constant ε could be built into the matrix A, however, the unique issue associated with the problem arises when this *viscosity coefficient* is small, therefore we are leaving it as a separate parameter. We will only present our results for some well known problems in figure 1.1, we will refer the interested reader to chapter 6 of the book [5], where the method and the reference problems are detailed.

Up until now, we have been looking at the 'quantity' of the convergence. We were taking a weak formulation

$$\int_\Omega A(x,u^{(n)}) \cdot \nabla u^{(n+1)} \cdot \nabla v + \int_\Omega q(x,u^{(n)})u^{(n+1)}v = \int_\Omega fv + \int_{\partial\Omega} pv \qquad \forall\, v \in H^1_D(\Omega)$$


(a) Test problem #1, $\varepsilon = 1/10$


(b) Test problem #1, $\varepsilon = 1/100$


(c) Test problem #2, $\varepsilon = 1/10$

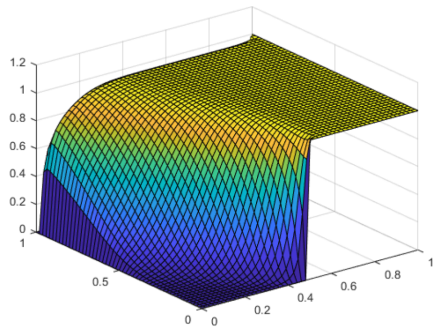

(d) Test problem #2, $\varepsilon = 1/100$

Figure 1.1: Two well-known test problems with viscosity coefficients $1/10$ (left) and $1/100$ (right)

then took some finite element subspace $V_h \subset H^1_D(\Omega)$, which only included the subspace of Courant (P1) elements last semester; this semester, we will also use bilinear (Q1) elements, but we have also implemented quadratic (P2) and biquadratic (Q2) elements. For more details on the construction of these, see e.g. [6]. Then the problem was reduced to numerical evaluation of the following integrals, and solving a simple linear algebraic equation for c:

$$A_n c = b_n,$$

where

$$A_n := \int_\Omega A(x,u^{(n)}) \cdot \nabla\varphi_j \cdot \nabla\varphi_i + \int_\Omega q(x,u^{(n)})\varphi_j\varphi_i$$

and

$$b_n := \int_\Omega f\varphi_i + \int_{\partial\Omega} p\varphi_i$$

are the individual elements of the matrix and the vector, with $\varphi_j,\varphi_i \in V_h$ basis function.

We can then increase n and keep solving the respective problems obtained, and after some time, we should be sufficiently close and can accept the last approximation $u^{(n+1)}$ as the solution. We could define some error for any given approximation $u^{(n)}$ and the exact (discretized) solution u, or two successitive approximations (the error defined by the Sobolev-norm is convenient, however one could also consider the max norm), and evaluate the convergence of the methods via these errors. We do not repeat our results here, and move on to this semester's findings.

2 Discrete nonnegativity principle

Sometimes we may be given a problem where we already have some qualitative information about the exact solution. The most prominent of these is the discrete nonnegativity- or nonpositivity principle. (It is easy to see why the two properties are equivalent.) On an analytical level, the former means that if we are given a problem

$$\mathcal{L}(u) = f$$

with some boundary conditions, where $\mathcal{L}$ is a differential operator, then for any right hand side function

$$f \geq 0 \qquad \text{we have} \qquad u \geq 0.$$

For more discussion on this property, confer [9].

It is clear that even if we have a numerical solution $u^{(n)}$ that is very close to the (discretized) exact solution in some norm but violates this property, it could be considered worse than a solution which may be further in norm, but maintains the nonnegativity principle. The first question is as follows: what conditions should we impose on our numerical method so that we can guarantee the property to hold true? In this project, however, we are investigating a second question: how accurate are these theoretical constraints (if they exist in this setting, at all)? First, we introduce our first problem, that will be a parabolic version of the problem introduced in [10]. We have the following initial-boundary value problem for the function $u : \Omega \times \mathbb{R}^+ \to \mathbb{R}$

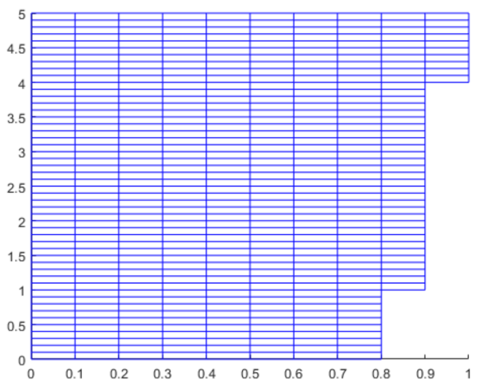
$$\begin{cases} \partial_t u - \operatorname{div}\left(A(u)\nabla u\right) = f & \text{in } \Omega \\ u(x,0) = 0 & \forall x \in \Omega, \\ u\big|_{\Gamma_D \times \mathbb{R}^+} = 0, \\ A(u)\partial_\nu u + \alpha(u - u_0)\big|_{\Gamma_N \times \mathbb{R}^+} = 0. \end{cases}$$

We proceed as usual with time-dependent problems: we describe some time levels t_j, where $t_j = j \cdot \tau$ with some time step. Then we account for the nonlinearity by freezing the coefficients and apply the one-point backward finite difference scheme for the time derivative, to obtain

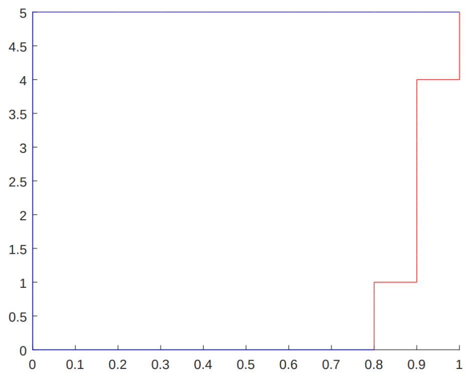
$$\frac{1}{\tau}(u^{(j+1)} - u^{(j)}) - \operatorname{div}\left(A(u^{(j)})\nabla u^{(j+1)}\right) = f,$$

then we can obtain the weak formulation as usual

$$\int\limits_\Omega A(u^{(j)}) \cdot \nabla u^{j+1} \cdot \nabla v + \int\limits_\Omega \frac{1}{\tau} u^{(j+1)} v + \int\limits_{\Gamma_N} \alpha u^{(j+1)} v = \int\limits_\Omega f v + \int\limits_\Omega \frac{1}{\tau} u^{(j)} v + \int\limits_{\Gamma_N} u_0 v.$$



(a) The discretization of the domain with Q1 elements



(b) Boundary conditions, Γ_D is denoted with blue, Γ_N is denoted with red.

Figure 2.1: The domain of the fuel transformer

This is not too hard to evaluate, all we have to do is calculate the stiffness and mass matrices for the given finite element (using the reference element method), and apply the boundary conditions, which can be done using a usual 'ghost-point' technique.

For our model, we will choose $A(u)$ to be a diagonal matrix, with both nonzero elements being

$$\frac{1+\mu_0(u-70)^2}{1+(u-70)^2},$$

and set f to be constant, $\alpha = 100$ and u_0 be a function that takes high values (328 Kelvins) near the top and the bottom of the domain, and vanishes everywhere else.

For this scheme, one can obtain (cf. [1]) for a given mesh size h a minimum step size

$$\tau_{\text{opt}} := \frac{1}{\frac{3}{\mu_0}\left(\left(\frac{h}{\mu_0}-\frac{\alpha}{2}\right)^2-\left(\frac{\alpha}{2}\right)^2\right)},$$

and if we invert the problem, for a given time step, we have a maximum mesh parameter

$$h_{\text{opt}} = \frac{\mu_0}{\frac{\alpha}{2}+\sqrt{\left(\frac{\alpha}{2}\right)^2+\frac{\mu_0}{3\tau}}}.$$

At first glance, this does not seem to be too restricting in the practical sense. We have all seen the Courant-Friedrichs-Levy condition (cf. [3]) that prescribes a maximum time step for a given grid size. This can render easy-to-implement explicit schemes unusable, since we would sometimes be constrained to imperceptible time steps, and long-term simulations would be computationally unfeasible. The consequence of the above condition is much more subtle, and definitely not irrelevant.

For example, whenever we are trying to solve unsteady problems in computational fluid dynamics, the first few moments always contain the most interest behavior, before the medium eventually starts settling into the stationary state. Hence, a logical timestepping method is taking small time steps close to $t = 0$, and then elongating the length of these steps to efficiently model the long-term behavior (see [5], chapter 10). Hence, if we have a premonition about the time steps we wish to take, we have to create a suitable mesh.

Similar considerations have to be taken into account when pricing assets. If we are pricing (long-dated) Bermudans or Americans with, say, daily fixing dates, we are forced to create our timestepping so that every day the option can be exercised is evaluated. In practice, even for options with longer interval between fixing dates we like to be more granular closer to the present day, before starting to take longer steps (although, since we are given terminal conditions, the times have to be inverted and when we are solving numerically, the time steps actually shrink over iterations). For more details, we recommend the book [4], especially chapter 7.

First, we have discretized the domain with $h = 0.1$ and set the time stepping parameter to $\tau = 0.01$. On the left side of the figure 2.2, we can see that in every time step, the minimum value among u is zero. At around $\tau = 0.004$, we start to violate the principle in the first few time steps, after which the minimum bounces back and the DNP is maintained. We see this on the right side in 2.2, albeit with an exaggerated τ value.

In theory, for this value of h, and the used parameters $\mu_0 := 0.8$ and $\alpha := 100$ we can calculate $\tau_{\text{opt}} = 0.067$ theoretical bound. In our experiments we have seen that we can take time steps as small as roughly 0.0047. For the other direction, we have taken $\tau = \frac{1}{16}$, which would give us a maximum mesh size $h = 0.1694$, but in practice, we could have the mesh size as high as $h = 1/3$ and not violate the principle. We can conclude that in this case, the estimation is not too accurate, as we have a factor of about 14 in the case when we fixed the mesh size, and while the factor is much better in the other case, it might just be that $h = 1/2$ is too coarse for these boundary conditions in itself.

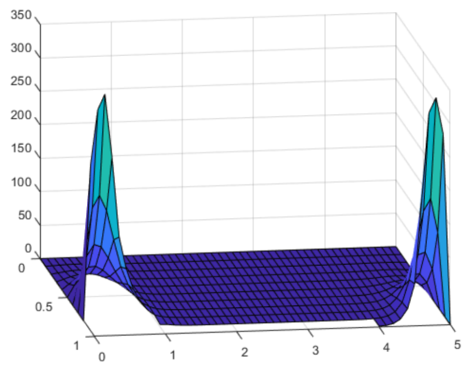
3 Irregular domains

Our second problem concerns the *Michaelis-Menten equation*, which is a reaction-diffusion-type equation in a (biological) cell, characterized by

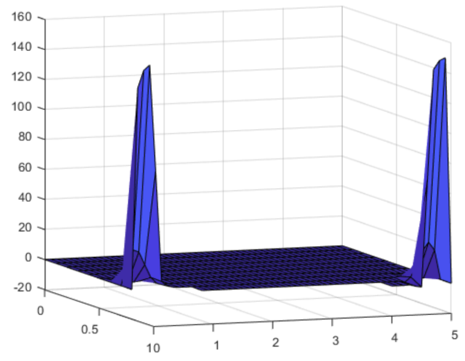
$$\begin{cases} \partial_t u - \mu_0 \Delta u + \dfrac{u}{1+\varepsilon u} = f & u : \Omega \times \mathbb{R}^+, \\ u(x,0) = 0 & \forall\, x \in \Omega, \\ u\big|_{\partial\Omega \times \mathbb{R}^+} = 0. \end{cases}$$

Since a square or a rectangular cell is not too realistic, we first have to find a way to create discretizations of more general domains. We have done this using the following algorithm, detailed in [2]:

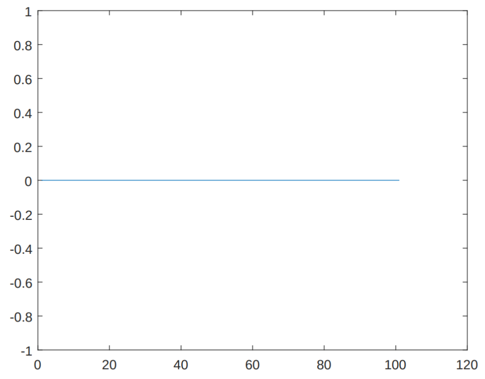
First, we need to somehow define the domain. The algorithm takes two inputs for this, one containing the fixed (boundary) vertices, and one containing connectivity data. Further, we have a parameter that determines the mesh size.



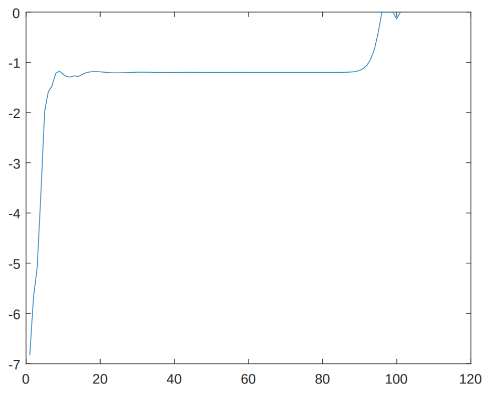
(a) The solution with $h = 0.1$, $\tau = 0.01$ after 5 time steps



(b) The solution with $h = 0.1$, $\tau = 0.001$ after 5 time steps



(c) The evolution of the minimum during timestepping, $\tau = 0.01$



(d) The evolution of the minimum during timestepping, $\tau = 0.001$

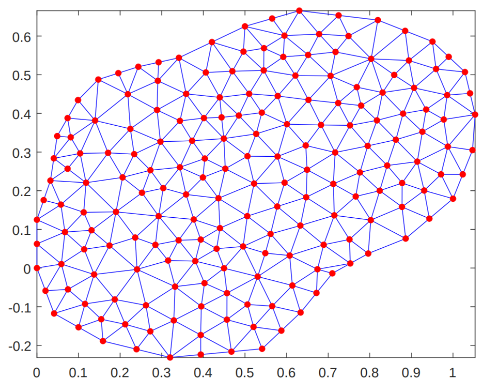
Figure 2.2: The discrete nonnegativity principle for two sets of parameters

Second, we dissect the boundary edges by adding vertices, so that no inner edge is longer than the prescribed value. With this, we have obtained a discretization of the boundary; our next step is adding points inside the domain. We place an underlying square grid onto the domain, and into every grid cell, we generate 5 random points. We then loop over the points, and choose ones that are at least $h/2$ distance away from each other, as well as throw out those that are too close to the boundary, since including these would almost always result in a triangulation of bad quality.

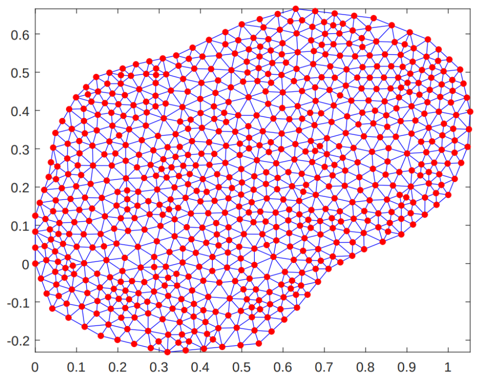
Next, we have to determine some way to connect these points with triangles. We do this with two considerations in mind: we want the triangles to be as close to equilateral, but also, make it so that as a consequence of adding a triangle, we will not be forced to add a bad quality triangle. That is, we start at an edge AB, choose the nearest k-many points C_j, where the triangle ABC_j does not contain another point. Then for these k-many triangles, we can calculate three scores: how close is ABC_j to an equilateral triangle, and for the edges AC_j and BC_j, calculate how close the triangle AC_jD_j and $BC_j\widehat{D}_j$ are to equilateral triangles (where D and $\widehat{D}$ are the closest nodes to the respective edges). Based on these three scores, we choose a vertex C_j and add the corresponding triangle to the triangulation.

Lastly, it is well known (cf. [5]) that the error on a given element can be estimated as the corresponding error on the reference element, divided by the sine of its smallest angle. That is, we wish to avoid creation triangulations that have 'tiny slivers', so we employ a post-smoothing iteration: as long as we deem necessary, we loop over all points, and move them to the barycenter of the polygon defined by its neighbors.

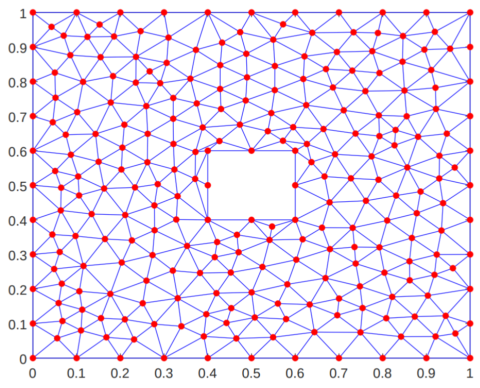
Before we move on, let us support the choice of the algorithm: we could have implemented a Delaunay-triangulation based algorithm, however, those algorithms can not triangulate concave domains (or at least, not without generating a triangulation on the convex hull of the domain and then removing the unnecessary parts). However, Lawson's algorithm (cf. [12]) allows us to flip edges so that our triangulation fulfills the Delaunay-property. In practice, our algorithm was only a few edge flips away from a Delaunay-triangulation, and is able to generate meshes for many domains.



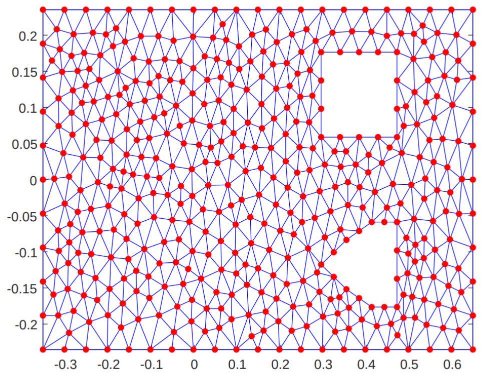
(a) Mesh #1: A cell



(b) Mesh #2: A cell with smaller mesh paramater



(c) Mesh #3: a square with a square hole



(d) Mesh #4: a wind tunnel with some objects inside

Figure 3.1: Various meshes

Here, if we are given a mesh size h_0 (so, the longest edge in the mesh has length h_0), then we can obtain a minimum step size (cf. [1])

$$\tau_{\text{opt}} := \frac{1}{\frac{12\cos(\alpha_0)\cdot\mu_0}{h_0} - 1},$$

and conversely, if we are given a time step τ, we get a maximum mesh size

$$h_{\text{opt}} := \left(\frac{12 \cos(\alpha_0) \cdot \mu_0}{1 + \frac{1}{\tau}} \right)^{1/2} .$$

We immediately see that this does not make any sense if α_0, the maximum angle among all triangles is greater than 90 degrees, therefore we have to make acute meshes.

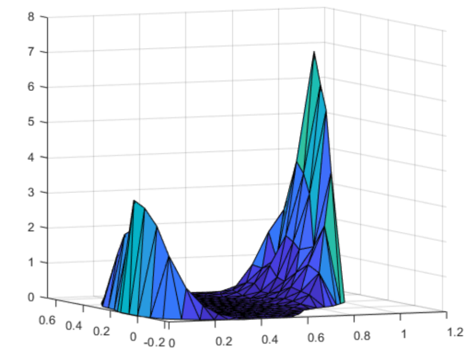
Unfortunately, only heuristics exist for such algorithms. An industry-standard satisfice is described in [7], although their algorithm tends to leave triangles that have angles close to right angles, making such meshes similarly useless for estimations. A good overview of the capabilities of such algorithms is given in the introduction of [11], where they claim that in practice, we can prescribe a minimum angle $\gamma = 41°$, and a maximum angle $\alpha = 81°$. Unfortunately, due to the difficulties arising during the implementation of such algorithms, we could not do a proper implementation, and instead relied on generating many meshes, and throwing away the ones that violate the angle condition, attaining a set of acute triangulations of our cell.

In this case, we have taken a look at three different mesh parameters: $h_1 = 0.015$, $h_2 = 0.03$, $h_3 = 0.6$. We have managed to generate meshes with maximum angles $\alpha_1 = 85.6754°$, $\alpha_2 = 83.3734°$ and $\alpha_3 = 83.6098°$. With the parameter $\mu_0 = 0.01$, this would result in the minimum time steps $\tau_1 = 0.0255$, $\tau_2 = 0.07$ and $\tau_3 = 0.37$. In practice, we have managed to get the time steps as small as $\widehat{\tau}_1 = 0.008$, $\widehat{\tau}_2 = 0.03$ and $\widehat{\tau}_3 = 0.18$, so our estimate is accurate only up to a factor of about two or three. Some results are presented in figure 3.2.

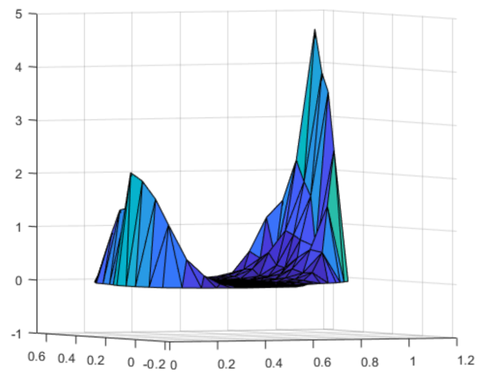
4 Future work

Lastly, we would like to point out that there is ample potential for future investigations. We have started work on a prismatic domain, which extends the scope of the method into three dimensions. Further, the introduction of P2 and Q2 elements allow for solving the Stokes- and Navier-Stokes problems, and since creating meshes for these consist of adding extra nodes to our already existing grids, the last remaining difficulties remain in modifying our matrix with extra elements to account for the new unknown, the pressure vector.

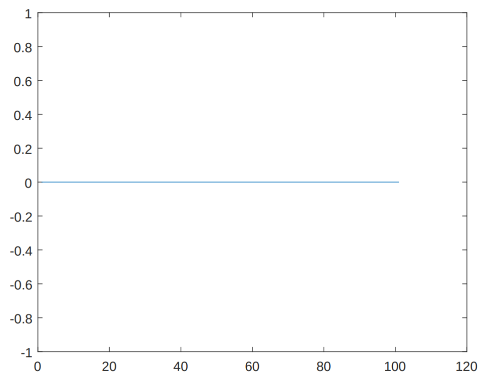
Anither aspect we intend to work on (as part of my thesis work) is finishing the mesh generation algorithms. We have already mentioned that we have not managed to implement algorithms that guarantee high-quality meshing for triangular elements, and further, we have not implemented a general mesh generation algorithm for quadrilateral elements. Lastly, one aspect we have not taken into account is the solution of the linear equation $Ac = b$. In structured meshes, we could use the multigrid method for fast convergence, however, that needed presumptions about the mesh. However, a generalization called *algebraic multigrid* not only replicates the greatness of the (geometric) multigrid algorithm, but in some cases, even exceeds it. Development and investigation of such an algorithm could be a part of future work as well.



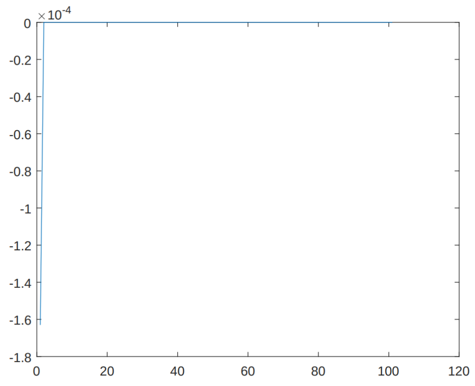
(a) The solution with $h = 0.06$, $\tau = 0.2$ after 5 time steps



(b) The solution with $h = 0.06$, $\tau = 0.1$ after 10 time steps



(c) The evolution of the minimum during timestepping, $\tau = 0.2$



(d) The evolution of the minimum during timestepping, $\tau = 0.1$

Figure 3.2: The discrete nonnegativity principle for two sets of parameters

References

[1] Bahlibi M. T., Karátson, J., Korotov, S. and András, S., Discrete non-negativity with computable conditions for some time-discretized parabolic finite elements problems – *in preparation*

[2] Cavendish, J. C. (1974). Automatic triangulation of arbitrary planar domains for the finite element method. *International Journal for Numerical Methods in Engineering, 8(4), 679-696.*

[3] Courant, R., Friedrichs, K., & Lewy, H. (1928). Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische annalen, 100(1), 32-74.*

[4] Clark, I. J. (2011). Foreign exchange option pricing: a practitioner's guide. *John Wiley & Sons.*

[5] Elman, H. C., Silvester, D. J., & Wathen, A. J. (2014). Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. *Oxford university press.*

[6] Ern, A., & Guermond, J. L. (2021). Finite elements (Vol. 72). *Berlin: Springer.*

[7] Erten, H., & Üngör, A. (2007, August). Computing Acute and Non-obtuse Triangulations. *In CCCG (Vol. 2007, pp. 205-208).*

[8] Faragó, I., & Karátson, J. (2002). Numerical solution of nonlinear elliptic problems via preconditioning operators: Theory and applications (Vol. 11). *Nova Publishers.*

[9] Faragó. I., Karátson, J., and Korotov, S., Discrete maximum principles for nonlinear parabolic PDE systems. *IMA Journal of Numerical Analysis, Oxford University Press 32(4), 1541-1573 (2012)*

[10] Hlavácek, I., Krizek, M., & Maly, J. (1994). On Galerkin approximations of a quasilinear nonpotential elliptic problem of a nonmonotone type. *Journal of Mathematical Analysis and Applications*, 184(1), 168-189.

[11] Khan, D., Yan, D. M., Wang, Y., Hu, K., Ye, J., & Zhang, X. (2018). High-quality 2D mesh generation without obtuse and small angles. *Computers & Mathematics with Applications, 75(2), 582-595.*

[12] Lawson, C. L. (1977). Software for C^1 Surface Interpolation. *Mathematical Software III (John R. Rice, editor).*