UPPSALA
UNIVERSITET

Advanced Numerical Methods

Stabilized Finite Element Methods

Authors:

Matilda Tiberg

matilda.tiberg.5927@student.uu.se

Uppsala

September 21, 2022

Contents

1 Introduction

In this project, the two-dimensional (2D) conservation law will be studied using the finite element method (FEM). The partial differential equation (PDE) is presented in equation (1). Here Ω is a fixed domain in $\mathbb{R}^2$ ad $\partial\Omega$ is defined as the boundary of the domain. In this project, homogeneous Dirichlet BC is used. The PDE is defined for time $T = (0, T]$ and has some known initial condition (IC) $u_0(\boldsymbol{x})$. The function $\boldsymbol{f} \in \mathcal{C}^1(\mathbb{R},\ \mathbb{R}^2)$ is a known flux term, which could be described as a term that moves things on the domain.

$$\begin{cases} \partial_t u + \nabla \cdot \mathbf{f}(u) = 0, & (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ u(\boldsymbol{x}, t) = 0, & \forall \boldsymbol{x} \in \partial\Omega, t \in (0, T] \\ u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \end{cases} \tag{1}$$

The first aim of this project is to implement the Galerkin method for FEM (GFEM) and then examine how different IC:s affects the correctness of the solutions. The convergence is studied and compared for two different IC:s: one smooth function and one sharp function.

The second aim of this project is to implement two different methods (Galerkin Least Squares and the Residual based artificial viscosity method) in order to stabilize the numerical solution. Both methods are being evaluated and compared to each other, as well to GFEM.

2 Galerkin FEM

In this part of the assignment, the 2D linear advection function, presented in equation (2), is being solved with the Galerkin method. Here, $\mathbf{b}$ is a vector field $\mathbf{b} = 2\pi(-x_2, x_1)$ called the convection term. The domain for the calculations consisting of a unit disc Ω. Homogeneous Dirichlet conditions are used at the boundary. The initial condition is set to either a smooth Gaussian function, or a circular step function of height 1.

$$\begin{cases} \partial_t u + \mathbf{b} \cdot \nabla u = 0, \ \ \forall \boldsymbol{x} \in \Omega, \ \ t \in (0,\ T] \\ u(\boldsymbol{x}, t) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega, \ \ t \in (0,\ T] \\ u(\boldsymbol{x}, 0) = u_0, \ \ \forall \boldsymbol{x} \in \Omega \end{cases} \tag{2}$$

The Crank-Nicolson method is used to fully discretize the problem in time, and the simulations are then being run for the maximal step sizes $h_{max} = [\frac{1}{4},\ \frac{1}{8},\ \frac{1}{16},\ \frac{1}{32}]$. The convergence is being examined, and the results from using two initial functions is then compared.

2.1 Discretization

To derive the GFEM formulation of equation 2, the PDE in said equation needs to be rewritten on weak form. To do this, find the test function $v \in V_0$, where $V_0 = H_0^1$. Applying the the inner product, denoted by $(\cdot, \cdot)$, to get the weak formulation. Then, we want to find $u \in V_0$, such that the following equation holds true:

$$(\partial_t u,\ v) + (\mathbf{b} \cdot \nabla u\ , v) = 0 \tag{3}$$

The triangulated mesh on the domain Ω is denoted as $\mathcal{T}_h = \{K\}$ and $\mathcal{N}_h$ is all the interior nodes on said mesh. Then the discrete trial space $V_{h,0}$ (defined in (4)), can be constructed, where $V_{h,0} \subset V_0$. Then, a discrete solution $u_h \in V_{h,0}$ can be found according to equation (5).

$$V_{h,0} = \{v :\ v \in C^0(\Omega),\ v|_K \in P_1(\kappa), \forall K \subset \tau_h, v = 0 \text{ on } \partial\Omega\} \tag{4}$$

$$(\partial_t u_h, v) + (\mathbf{b} \cdot \nabla u_h, v) = 0 \tag{5}$$

By using the definition $u_h = \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j(t)\varphi_j(\boldsymbol{x})$, equation (5), can be rewritten as equation (6), here on integral-from instead of inner product-notation.

$$\begin{aligned} 0 &= \int_\Omega \frac{\partial u_h}{\partial t} v \ d\boldsymbol{x} + \int_\Omega \mathbf{b} \cdot \nabla u_h v \ d\boldsymbol{x} \\ &= \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_j(t)}{dt} \int_\Omega \varphi_j(\boldsymbol{x})\varphi_i(\boldsymbol{x}) \ d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j(t) \int_\Omega \mathbf{b} \cdot \nabla \varphi_j(\boldsymbol{x})\varphi_i(\boldsymbol{x}) \ d\boldsymbol{x} \\ &= \mathrm{M}\frac{d\boldsymbol{\xi}(t)}{dt} + C\boldsymbol{\xi}(t) \end{aligned} \tag{6}$$

where M is the mass matrix $\mathrm{M} = \int_\Omega \varphi_j(\boldsymbol{x})\varphi_i(\boldsymbol{x}) d\boldsymbol{x}$, and C is the convection matrix $\mathrm{C} = \int_\Omega \mathbf{b} \cdot \nabla \varphi_j(\boldsymbol{x})\varphi_i(\boldsymbol{x}) d\boldsymbol{x}$. The problem now consists of an ODE of first degree, with initial condition depending on the chosen problem.

$$\begin{cases} \mathrm{M}\frac{d\boldsymbol{\xi}(t)}{dt} + C\boldsymbol{\xi}(t) = 0 \\ \boldsymbol{\xi}(0) = u(\boldsymbol{x}, 0) \end{cases} \tag{7}$$

To discretize the ODE in time, the Crank-Nicolson scheme is applied, which is a second-order accurate implicit FDM. The general case is shown in equation (8).

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = \frac{1}{2}[F_i^{n+1} + F_i^n] \tag{8}$$

Applying this on equation (7), it results in the derivation seen in equation (9), where k is the time step.

$$\begin{aligned} 0 &= M\frac{\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n}{k} + C\frac{\boldsymbol{\xi}^{n+1} + \boldsymbol{\xi}^n}{2} \\ &= 2M(\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n) + kC(\boldsymbol{\xi}^{n+1} + \boldsymbol{\xi}^n) \\ &= (kC + 2M)\boldsymbol{\xi}^{n+1} + (kC - 2M)\boldsymbol{\xi}^n \end{aligned} \tag{9}$$

This results in the fully discrete equation, shown in equation (10). When setting the time step k, the CFL-condition is set to $CFL = 0.5$, thus setting the value of the time step to $k = CFL\frac{h_{max}}{||\mathbf{b}||}$, which can be written as $k = CFL\frac{h_{max}}{\max\left(\sqrt{b^{(1)2} + b^{(2)2}}\right)}$. This ensures numerical stability.

$$\boldsymbol{\xi}^{n+1} = \Big[2M + kC\Big]^{-1}\Big[2M - kC\Big]\boldsymbol{\xi}^n \tag{10}$$

2.2 Implementation and Results

2.2.1 Gaussian Function as IC

In this part, the initial value of equation (2) is set to equation (11). This is a smooth Gaussian function, where $(x_1, x_2) \in \Omega$, $(x_1^0,\ x_2^0) = (0.3,\ 0)$ and $r_0 = 0.25$ determines the placement and size of the said function. When time stepping, the convection coefficient $\mathbf{b} = 2\pi[-x_2,\ x_1]$ makes this function circulate counter clockwise around the domain Ω. When $T = 1$, the function has made one full rotation around the circular domain, and thus the initial function is the same as the analytical function at $T = 1$. This is later used to calculate the error.

$$u_0(\mathbf{x}, 0) = \frac{1}{2}\left(1 - \tanh\left(\frac{(x_1 - x_1^0)^2 + (x_2 - x_2^0)^2}{r_0^2} - 1\right)\right) \tag{11}$$

Four different maximal mesh sizes were used when solving the problem: $h_{max} = [\frac{1}{4},\ \frac{1}{8},\ \frac{1}{16},\ \frac{1}{32}]$. In Figure 1 and 2 the results for $h_{max} = \frac{1}{8}$ respectively $h_{max} = \frac{1}{16}$ can be seen. When comparing the coarser mesh to the exact solution, it can be seen that there occurred oscillations in the numerical solution where the analytical solution is completely flat. In the finer mesh, there still occurred some oscillations, although they were much smaller. A top view of both numerical solutions is shown in Figure 3-4 to better show the oscillations.

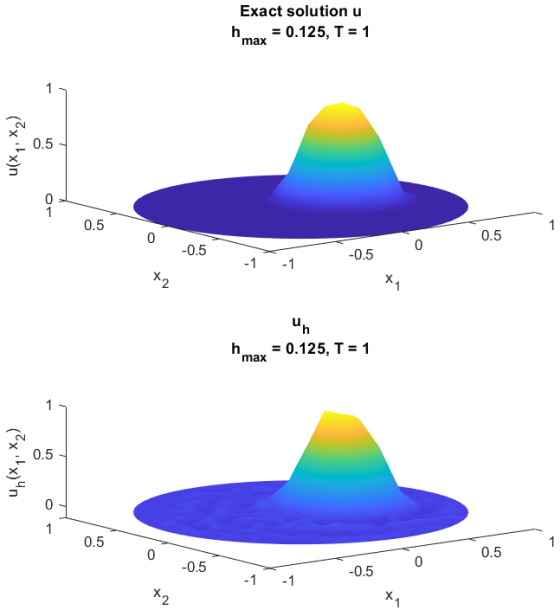

Figure 1: *The numerical solution with the smooth initial function, using* $h_{max} = 1/8$. *Analytical solution is shown for the same set up.*

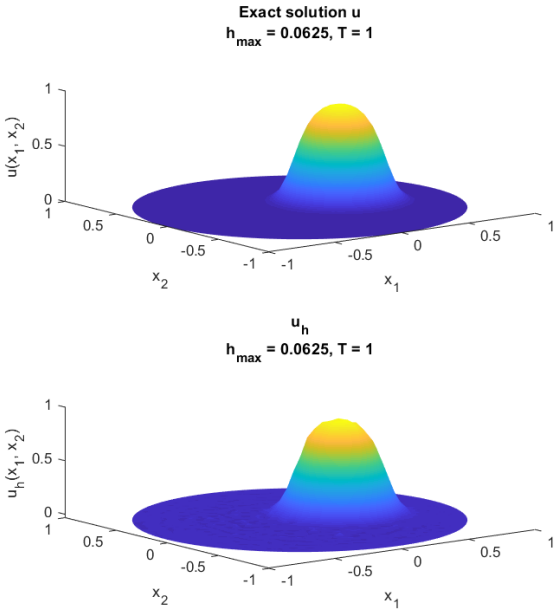

Figure 2: *The numerical solution with the smooth initial function, using* $h_{max} = 1/16$. *Analytical solution is shown for the same set up.*

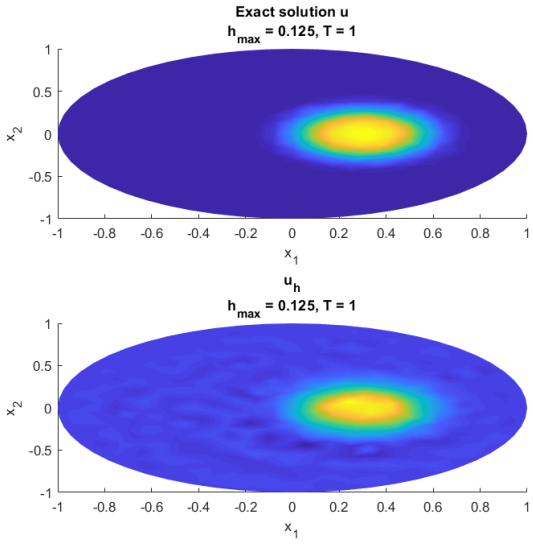

Figure 3: *The numerical solution with the smooth initial function, using* $h_{max} = 1/8$. *Analytical solution is shown for the same set up. Shown from top view.*

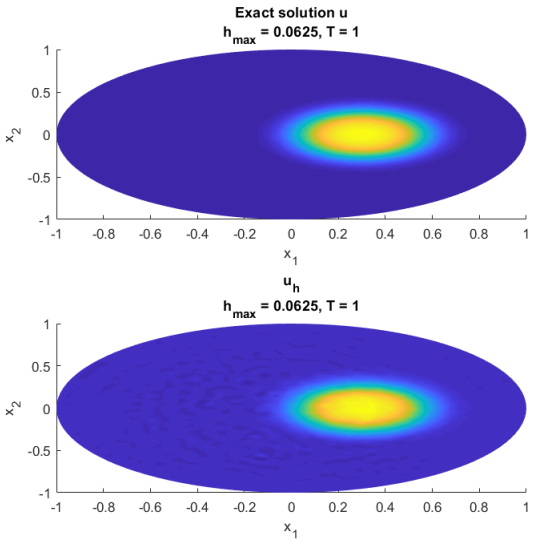

Figure 4: *The numerical solution with the smooth initial function, using* $h_{max} = 1/16$. *Analytical solution is shown for the same set up. Shown from top view.*

To calculate the error, the $L^2(\Omega)$-norm was used, calculated as $L2E = \sqrt{e^T M e}$, where M is the mass matrix, and e is the error defined as the difference between the analytical solution and the numerical solution: $e = u - u_h$. The convergence q between each h_{max} could then be calculated as stated in equation (12).

$$q_i = \frac{\log(L2E_i - L2E_{i-1})}{\log(h_{max,i} - h_{max,i-1})}, \qquad i = [2, 5] \tag{12}$$

The results for the L^2-norm of the error, as well as the convergence q are shown in Table 1. It can be seen that the error is decreasing steadily with an increasing h_{max}, and q reaches a maximum of $q = 1.8$. When plotting the error and fitting a line with MATLABs `polyfit`, seen in Figure 5, the convergence rate can be determined to $P = 1.67$, which is reasonable since the Galerkin method is a second order accurate method.

Table 1: *The calculated error L2E for each h_{max} and the computed convergence q. L2E is calculated as the L^2-norm, and q as the interpolated slope of L2E between each h_{max}.*

h_{max}	**L2E**	**q**
1/4	9.963E(-2)	-
1/8	3.455E(-2)	1.528
1/16	9.884E(-3)	1.806
1/32	3.161E(-3)	1.645
P	1.67	

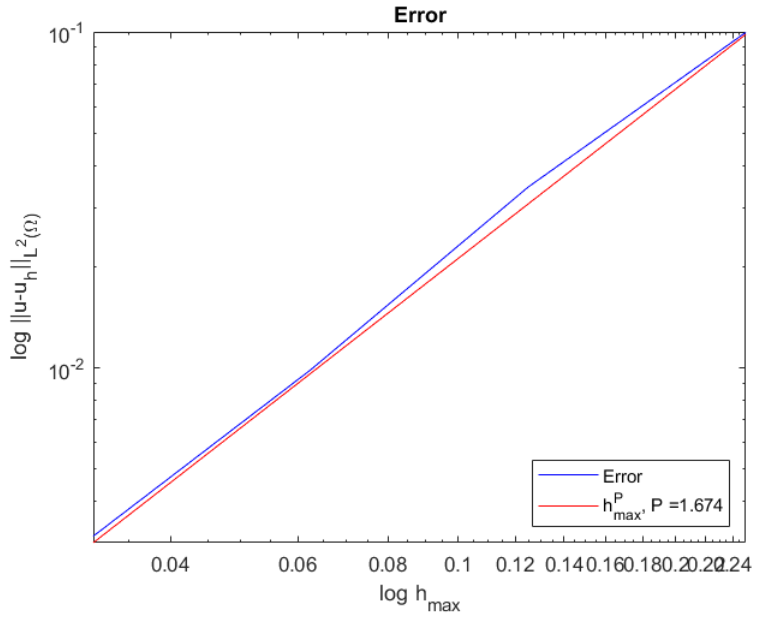

Figure 5: *The calculated $L^2(\Omega)$-norm of the error for Problem 1.1. Here compared to a line of slope $h_{max}^{1.674}$, which matches the average convergence rate.*

2.2.2 Step Function as IC

In this part, the initial value of equation (2) is set to equation (13). This function could be described as a circular unit step function, where $(x_1, x_2) \in \Omega$, $(x_1^0,\ x_2^0) = (0.3,\ 0)$ and $r_0 = 0.25$ determines the placement and radius of said function. When time stepping, the convection coefficient $\mathbf{b} = 2\pi[-x_2,\ x_1]$ makes this function circulate counter clockwise around the domain Ω. When $T = 1$, the function has made one full

rotation around the circular domain, and thus the initial function is the same as the analytical function at $T = 1$. This is later used to calculate the error.

$$u_0(\boldsymbol{x}, 0) = \begin{cases} 1, & \text{if } (x_1 - x_1^0)^2 + (x_2 - x_2^0)^2 \leq r_0^2 \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Four different maximal mesh sizes were used when solving the problem: $h_{max} = [\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}]$. In Figure 6 and 7 the results for $h_{max} = \frac{1}{8}$ respectively $h_{max} = \frac{1}{16}$ can be seen. When comparing the numerical solutions to the analytical solution, it can be seen that, in comparison to the previous section, FEM performs significantly worse when having a step function as initial function. There are lot of oscillations in both the coarser and finer meshes, as better demonstrated in the top views, presented in Figure 8-9. As seen, the analytical solution is not close to being circular for the coarser mesh, but this is due to the sparse nodes when using such a great value for h_{max}. The shape is more circular for the finer mesh, although still a bit rough. When plotting the solutions from a more angled view for the finer mesh, see Figure 10, it can be seen that the numerical solution does not have a flat peak, but rather a hole in the middle due to the instabilities.

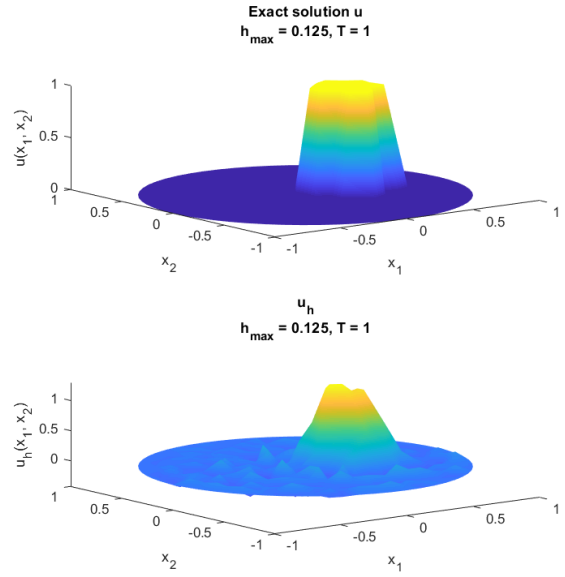

Figure 6: *The numerical solution with the sharp initial function, using $h_{max} = 1/8$. Analytical solution is shown for the same set up.*

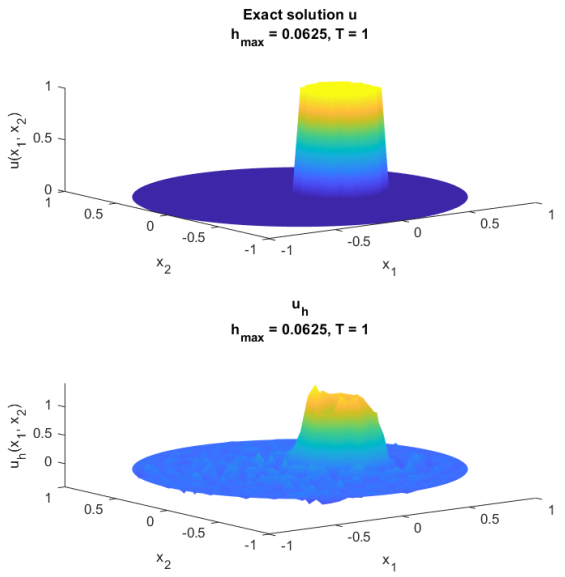

Figure 7: *The numerical solution with the sharp initial function, using $h_{max} = 1/16$. Analytical solution is shown for the same set up.*

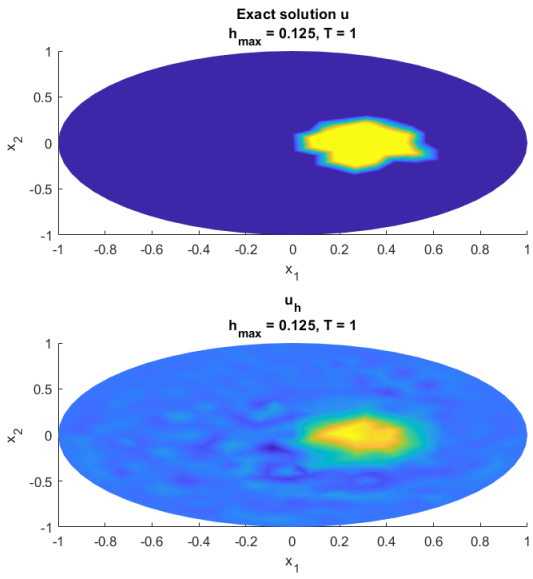

Figure 8: *The numerical solution with the sharp initial function, using* $h_{max} = 1/8$. *Analytical solution is shown for the same set up. Shown from top view.*

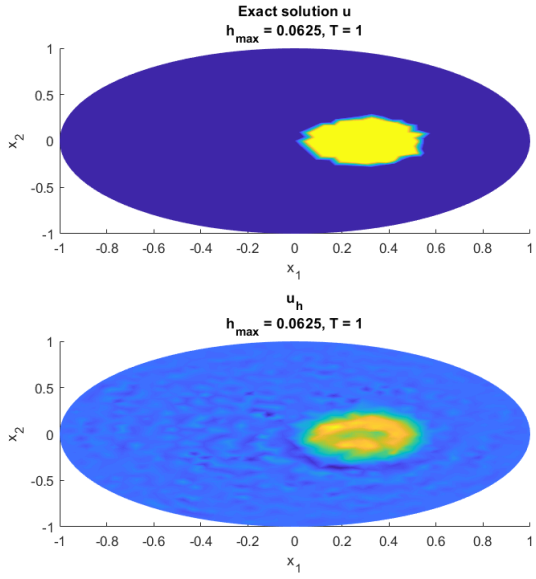

Figure 9: *The numerical solution with the sharp initial function, using* $h_{max} = 1/16$. *Analytical solution is shown for the same set up. Shown from top view.*

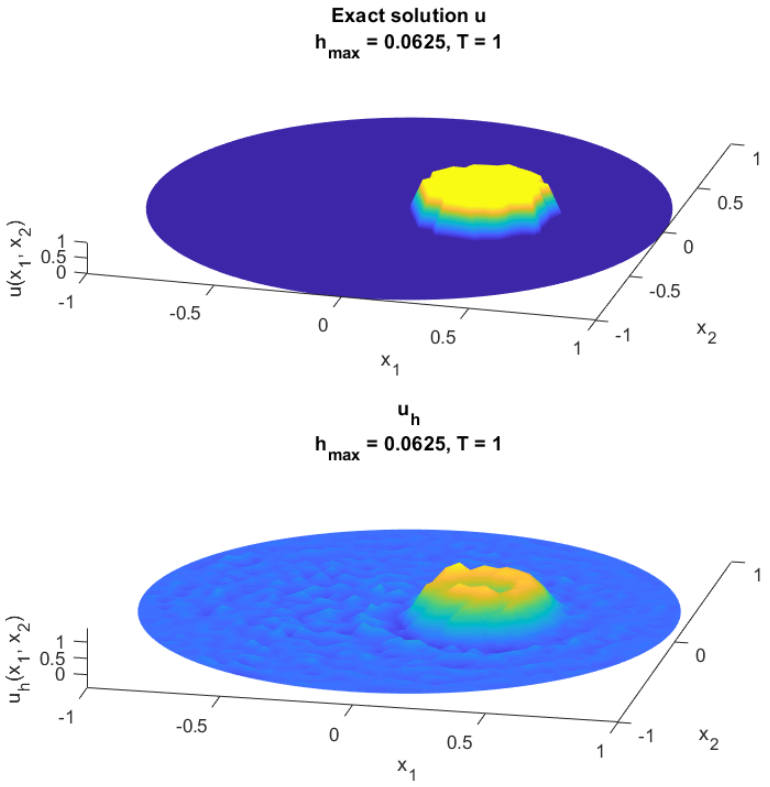

Figure 10: *The angled view for the solutions using* $h_{max} = \frac{1}{16}$ *at* $T = 1$ *using a step function as initial condition.*

The $L^2(\Omega)$-norm is calculated in the same way as for the smooth function: $L2E = \sqrt{e^T M e}$, and the convergence q is calculated as in equation (12). The results are presented in Table 2. As expected from the plots above, the normed error is great and the convergence q is very low. When fitting a line to the calculated error, the convergence rate $P = 0.223$ is achieved, as demonstrated in Figure 11. Since FEM has a convergence rate of 0.5 for discontinuous problems, the resulting convergence is somewhat low. As seen in the table, the q value does seem to increase in rate that makes the result tend to a more reasonable convergence rate for an increased value of h_{max}. However, even though that the convergence rate tends to the real numerical value, this method is still not suitable for this kind of problem since a lot of spurious oscillations are introduced to the solution.

Table 2: *The calculated error EnE for each* h_{max} *and the computed convergence* q*. EnE is calculated as the energy norm, and* q *as the slope of EnE between each* h_{max}*.*

h_{max}	**EnE**	**q**
1/4	0.1662	-
1/8	0.1513	0.1355
1/16	0.1298	0.2208
1/32	0.1045	0.3132
P	0.223	

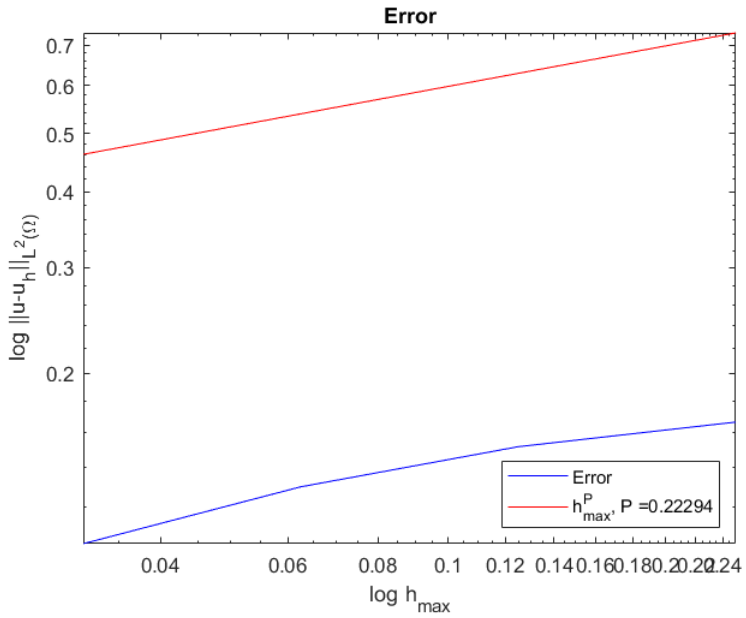

Figure 11: *The calculated $L^2(\Omega)$-norm of the error for Problem 1.1. Here compared to a line of slope $h_{max}^{0.223}$, which matches the average convergence rate.*

2.3 Discussion

When using the Gaussian function as the initial condition, the performance was significantly better compared to when using a step function. The error became smaller for both problems with an reduced value of h_{max}, however, for the step function the error was not reduced in satisfying rate: only achieving a convergence rate of $P = 0.223$, as compared to the smooth function, were a convergence of $P = 1.67$ was achieved. This could also be concluded from the resulting plots, where the oscillations disappeared with a reduced size of h_{max} at contrary to the step function where the oscillations remained.

All these results indicates that sharp edges creates instabilities for FEM. This is due to the fact that the Galerkin space discretization is unstable for the flux term, since the scheme is equivalent to the central difference approximation. This is problematic, since the non-smoothness causes singularities due to the non-existence of a derivative at the sharp corners, thus some kind of stability factor needs to be introduced.

3 Stabilized FEM

From before it could be seen that the ordinary GFEM method resulted in oscillations for both IC:s, especially for the step function. As concluded in the previous discussion, this is due to the fact that GFEM is a central difference scheme, which does not handle sharp functions well, nor convection dominated problems. In order to surpass this, different manipulations could be done to the numerical solution. In this section, two different methods will be imposed to add an artificial diffusion factor to the problem in order to smoothen the problem while still keeping the accuracy. These two methods are the *Galerkin Least Square* method (GLS) and the *Residual based artificial viscosity* method (RV-method).

3.1 Galerkin Least Squares

The GLS is obtained by combining the accuracy of GFEM and robustness of the Least Square method in order to minimize oscillations while still maintaining good convergence. This is done by introducing a parameter δ and replacing the test function v by the expression $v + \delta Lv$. Here L is the differential operator of the given PDE. The GLS method then becomes as following:

$$(Lu, v) + \delta(Lu, Lv) = (f, v) + \delta(f, Lv) \tag{14}$$

3.1.1 Discretization

The differential operator L for the PDE in equation (2) is given by $L = \partial_t + \mathbf{b} \cdot \nabla$. Thus, the PDE can be written on the form $Lu = 0$. Using the expression for GLS in (14), we find the weak form as follows:
Find $u \in V_0 = H_0^2$ such that

$$(\partial_t u + \mathbf{b} \cdot \nabla u,\ v) + \delta(\partial_t u + \mathbf{b} \cdot \nabla u,\ \partial_t v + \mathbf{b} \cdot \nabla v) = 0, \qquad \forall v \in V_0 \tag{15}$$

To spatially discretize, find $u_h \in V_{h,0}$ (defined in equation (4), and $V_{h,0} \subset V_0$), such that

$$(\partial_t u_h + \mathbf{b} \cdot \nabla u_h,\ v) + \delta(\partial_t u_h + \mathbf{b} \cdot \nabla u_h,\ \partial_t v + \mathbf{b} \cdot \nabla v) = 0 \qquad \forall v \in V_{h,0} \tag{16}$$

By using the definition $u_h = \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j(t)\varphi_j(\boldsymbol{x})$, where $\varphi_j(\boldsymbol{x})$ are the pieceswise linear hat functions, equation (16) can be written as the following below. Here, the term containing the time derivative of v can be omitted.

$$\begin{aligned}
0 &= \int_\Omega \frac{\partial u_h}{\partial t} v \ d\boldsymbol{x} + \int_\Omega \mathbf{b} \cdot \nabla u_h v \ d\boldsymbol{x} + \delta \int_\Omega \partial_t u_h \mathbf{b} \cdot \nabla v \ d\boldsymbol{x} + \delta \int_\Omega \mathbf{b} \cdot \nabla u_h \mathbf{b} \cdot \nabla v \ d\boldsymbol{x} \\
&= \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_j}{dt} \int_\Omega \varphi_j \varphi_i \ d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j \int_\Omega \mathbf{b} \cdot \nabla \varphi_j \varphi_i \ d\boldsymbol{x} + \\
&\qquad + \delta \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_j}{dt} \int_\Omega \varphi_j \mathbf{b} \cdot \nabla \varphi_i \ d\boldsymbol{x} \ + \delta \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j \int_\Omega \mathbf{b} \cdot \nabla \varphi_j \mathbf{b} \cdot \nabla \varphi_i \ d\boldsymbol{x} \\
&= M \frac{d\boldsymbol{\xi}}{dt} + C\boldsymbol{\xi} + \delta C^T \frac{d\boldsymbol{\xi}}{dt} + \delta S_D \boldsymbol{\xi}
\end{aligned} \tag{17}$$

The mass matrix M and convection matrix C are defined as previously, and S_D is the streamline diffusion matrix defined as $S_D = \sum_j \xi_j \int_\Omega \mathbf{b} \nabla \varphi_j \nabla \varphi_i d\boldsymbol{x}$. The matrix C^T is the transpose of the convection matrix C. Thus, the following ODE is obtained:

$$\begin{cases} (M + \delta C^T)\frac{d\boldsymbol{\xi}}{dt} + (C + \delta S_D)\boldsymbol{\xi} = 0 \\ \boldsymbol{\xi}(0) = u(\boldsymbol{x}, 0) \end{cases} \tag{18}$$

Applying the Crank-Nicolson scheme, from equation (8), on the ODE to discretize in time, where k denotes the temporal step size, we then get the fully discretized formulation:

$$\boldsymbol{\xi}^{n+1} = \Big[\frac{1}{k}(M + \delta C^T) + \frac{1}{2}(C + \delta S_D)\Big]^{-1}\Big[\frac{1}{k}(M + \delta C^T) - \frac{1}{2}(C + \delta S_D)\Big]\boldsymbol{\xi}^n \tag{19}$$

3.2 Proving Stability

Before, we have defined $L = \partial_t + \mathbf{b} \cdot \nabla$, thus getting $Lu = 0$. Assuming that the field is divergence free, i.e. $\nabla \cdot \mathbf{b}$, we can apply GLS and find $u_h \in V_h$ such that

$$(\partial_t u_h + \mathbf{b} \cdot \nabla u_h,\ v) + \delta(\partial_t u_h + \mathbf{b} \cdot \nabla u_h,\ \partial_t v + \mathbf{b} \cdot \nabla v) = (0,\ v) + \delta(0,\ \partial_t v + \mathbf{b} \cdot \nabla v) \qquad \forall v \in V_h \tag{20}$$

where the RHS of above equation equals to zero.

Using the relation $(\partial_t u_h,\ u_h) = \frac{1}{2}\partial_t ||u_h||^2$, and the definition of L together with $(\mathbf{b} \cdot \nabla u_h,\ u_h) = 0$ (due to the conditions of no divergence) the energy estimate follows as:

$$\frac{1}{2}\partial_t ||u_h||^2 + \delta ||Lu||^2 = 0 \tag{21}$$

Now integrating over time $t \in [0, T]$:

$$\int_0^T \frac{1}{2}\partial_t ||u_h||^2 + \delta ||Lu||^2 \, dt = 0 \tag{22}$$

This gives equation (23) which states that the GLS method applied to this kind of is energy conservative: i.e. no energy will be lost while time stepping. By definition, this means stability since all terms are bounded.

$$||u_h(T)||^2 + \delta \int_0^T ||Lu_h(\cdot, t)||^2 dt = ||u_0||^2 \tag{23}$$

3.3 Results GLS

As before, a maximal spatial step size of $h_{max} = [\frac{1}{4},\ \frac{1}{8},\ \frac{1}{16},\ \frac{1}{32}]$ was used, and the two different initial functions in equation (11) and (13). All simulations was run to $T = 1$ in order to enable comparison to the analytical solution. The codes can be found in Appendix B. In this part, $\delta = \frac{1}{2}h/||\mathbf{b}||_\infty$ was used.

The numerical results for the smooth IC, using $h_{max} = \frac{1}{8}$ and $h_{max} = \frac{1}{16}$, are shown in Figure 12-13. When comparing the numerical and the analytical solutions, it can be seen that the numerical solution is somewhat squished as compared to the analytical solution, but in general, they are very similar. When comparing these results to the results for GFEM with smooth IC (Figure 1-2), it can be seen that the oscillations at the flat surface has disappeared. This is due to the diffusive properties that the least squares-terms brings to the solution.

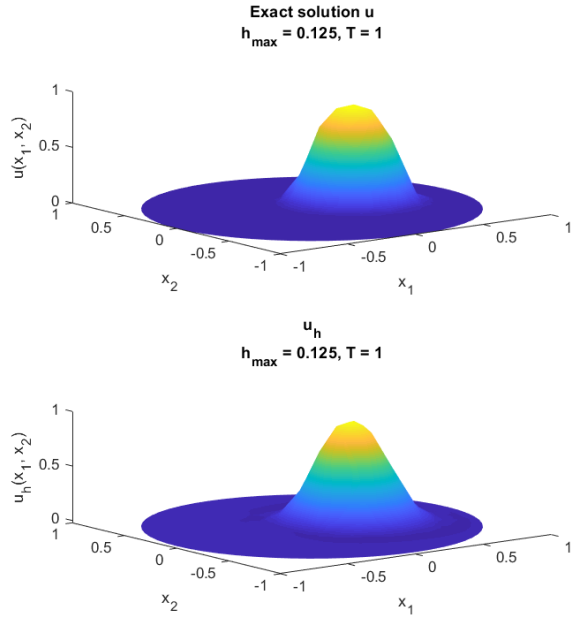

Figure 12: *The results for $h_{max} = \frac{1}{8}$ using GLS and smooth IC. Compared to the analytical solution.*

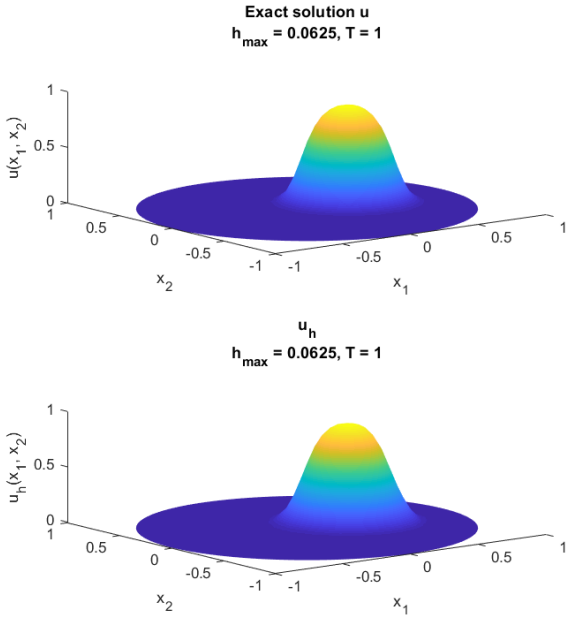

Figure 13: *The results for $h_{max} = \frac{1}{16}$ using GLS and smooth IC. Compared to the analytical solution.*

When applying the GLS-method to the step IC the diffusion affects the results in a greater way: the oscillations are greatly reduced and the numerical solution looks more like a cylindrical step than the GFEM solutions presented in Figure 6-7. The resulting distribution is a bit smoothened, which is more apparent for the coarser mesh. The numerical solution for the finer mesh is a bit more true to the analytical solution, but it could no longer be considered to be a step function due to the smoothening.

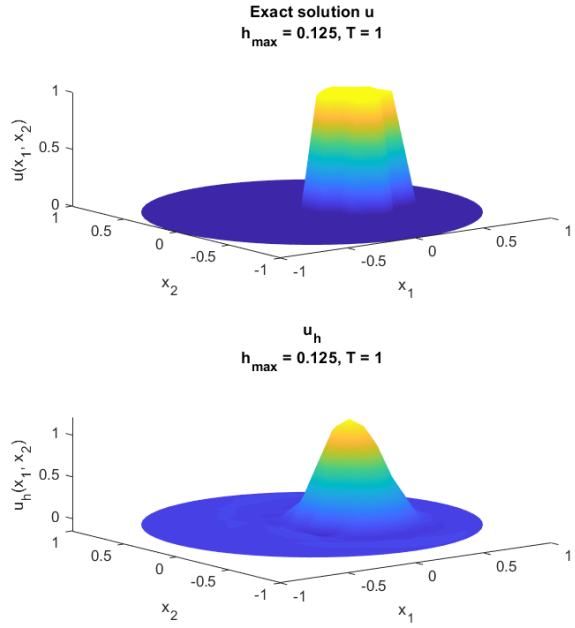

Figure 14: *The results for $h_{max} = \frac{1}{8}$ using GLS and step IC. Compared to the analytical solution.*

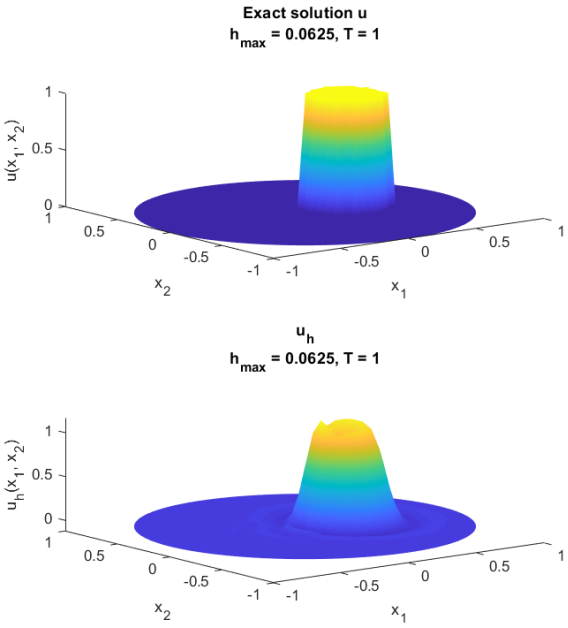

Figure 15: *The results for $h_{max} = \frac{1}{16}$ using GLS and step IC. Compared to the analytical solution.*

In Table 3, the L^2-normed error and the convergence rates are presented. When comparing to the GFEM results in Table 1-2, it can be seen that the convergence rate for GLS is higher than GFEM for this type of problems. This is reasonable when comparing to the plots, since all the spurious oscillations got diffused, which was expected since an dissipation term was introduced. Fitting a line to the error, a convergence rate of $P = 1.83$ was achieved for the smooth IC. The step IC achieved a convergence rate of $P = 0.302$. The error plots are presented in Figure 16-17.

Table 3: *The calculated error L2E for each h_{max} and the computed convergence q. L2E is calculated as the L^2-norm, and q as the interpolated slope of L2E between each h_{max}. P is the average convergence rate.*

	Smooth IC		*Step IC*	
h_{max}	**L2E**	**q**	**L2E**	**q**
1/4	0.1014	-	0.1531	-
1/8	3.284E(-2)	1.627	0.1215	0.3329
1/16	1.038E(-2)	1.662	9.970E(-2)	0.2857
1/32	2.171E(-3)	2.257	8.146E(-2)	0.2915
P	1.83		0.302	

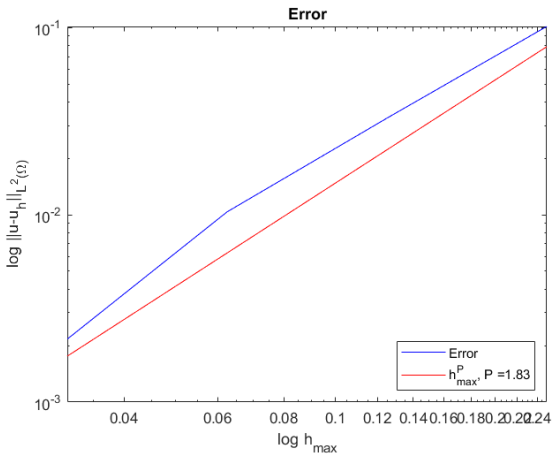

Figure 16: *The calculated $L^2(\Omega)$-norm of the error for Gaussian IC, using GLS. Here compared to the average convergence rate.*

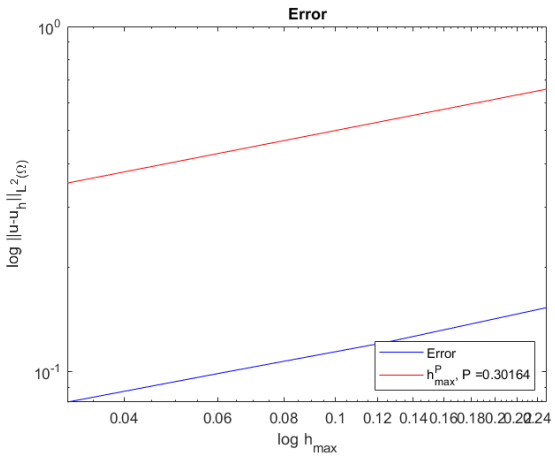

Figure 17: *The calculated $L^2(\Omega)$-norm of the error for step IC, using GLS. Here compared to the average convergence rate.*

3.4 Residual Based Artificial Viscosity

This method stabilizes the FEM-scheme by introducing an artificial dissipation term, shown in equation (24). This stabilizes works by altering the central difference scheme, to avoid having problems at sharp edges, which the ordinary GFEM-scheme experiences. As ε tends to zero, the RV-method gets closer to GFEM.

$$\varepsilon \sim \min\left(c_{vel} h_K \beta_K,\ c_{RV} h_K^2 \frac{||R_{RV}||_{\infty,K}}{||\xi^n - \overline{\xi}^n||_{\infty,\Omega}}\right) \tag{24}$$

3.4.1 Discretization

When discretizing with this method, the steps are similar to before, however, one extra term is added:

$$(\partial u_h,\ v) + (\mathbf{b}\cdot\nabla u_h,\ v) + (\varepsilon_h \nabla u_h,\ v) = 0 \qquad \forall v \in V_{h,0} \tag{25}$$

$$\sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_j}{dt} \int_\Omega \varphi_j \varphi_i \ d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j \int_\Omega \mathbf{b}\cdot\nabla\varphi_j \varphi_i \ d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j \int_\Omega \varepsilon_h \nabla\varphi_j \nabla\varphi_i \ d\boldsymbol{x} = 0 \tag{26}$$

Since ε_h is both temporally and spatially dependent, it cannot be moved out of the integral. Hence, the stiffness matrix is slightly modified to $R_V = \sum_j \xi_j \int_\Omega \varepsilon_h \nabla\varphi_j \nabla\varphi_i d\boldsymbol{x}$, and thus needs to be assembled at every time step. The semi-discrete problem is then achieved:

$$M\frac{d\boldsymbol{\xi}}{dt} + C\boldsymbol{\xi} + R_V\boldsymbol{\xi} = 0 \tag{27}$$

Applying the Crank-Nicolson scheme to fully discretize the problem reults in the following ODE:

$$\boldsymbol{\xi}^{n+1} = \Big[2M + k(C + R_V)\Big]^{-1}\Big[2M - k(C + R_V)\Big]\boldsymbol{\xi}^n \tag{28}$$

R_V contains the residual term $R(u_h) = \partial_t u_h + \mathbf{b}\cdot\nabla u_h$ and thus needs to be updated at every time step. However, the term $\mathbf{b}\cdot\nabla u_h$ does not live in the FEM-space, and thus $R(u_h)$ needs to be found using L^2-projection:

Find $R(u_h) \in V_h$ such that

$$\begin{aligned} R_{n+1}(u_h) &= \left(\frac{u_h^{n+1} - u_h^n}{k},\ v \right) + (\mathbf{b} \cdot \nabla u_h^n,\ v) \qquad \forall v \in V_h \\ &= \frac{1}{k} M(\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n) + C\boldsymbol{\xi}^{n+1} = \varphi_i \end{aligned} \tag{29}$$

Using the relation $R(u_h) = \eta\varphi_i$, we get the residual η as the following:

$$\eta = M^{-1}\left[\frac{1}{k} M(\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n) + C\boldsymbol{\xi}^{n+1}\right] \tag{30}$$

3.5 Results RV-method

As before, a maximal spatial step size of $h_{max} = [\frac{1}{4},\ \frac{1}{8},\ \frac{1}{16},\ \frac{1}{32}]$ was used, and the two different initial functions in equation (11) and (13). All simulations was run to $T = 1$ in order to enable comparison to the analytical solution. The codes can be found in Appendix C. Something worth to mention is that the runtime for this method was extensive in comparison to all other methods.

Figure 18 and 19 presents the numerical results for $h_{max} = \frac{1}{8}$ and $h_{max} = \frac{1}{16}$ for the RV-method applied on smooth IC. It can be seen that the coarser mesh is a bit more flattened than the finer mesh. When comparing the results for the RV-method with the results for GFEM, shown in Figure 1-2, it can be seen that there are fewer oscillations surrounding the peak when using the RV-method. This difference is more visible when comparing the coarser meshes.

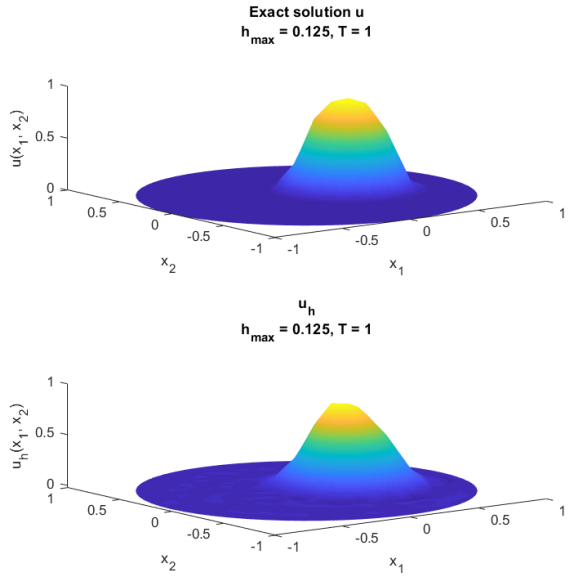

Figure 18: *The results for $h_{max} = \frac{1}{8}$ using the RV-method and smooth IC. Compared to the analytical solution.*

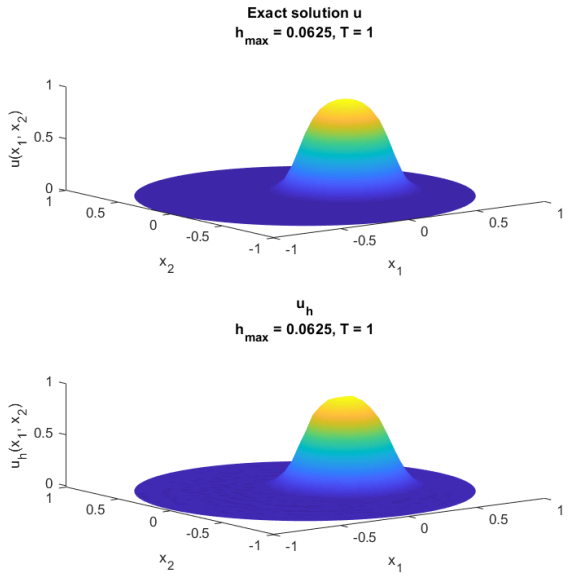

Figure 19: *The results for $h_{max} = \frac{1}{16}$ using the RV-method and smooth IC. Compared to the analytical solution.*

Figure 20 and 21 presents the numerical results for $h_{max} = \frac{1}{8}$ and $h_{max} = \frac{1}{16}$ for the RV-method applied on the step IC. It can be seen that the coarser mesh is significantly squished, and reminds more of a Gaussian

distribution than a step function. For the finer mesh, the results looks more like the analytical solution, but still has smoothened edges. However, when comparing to the GFEM solutions (Figure 6-7), there is a significant difference in the results for both the coarse and fine meshes: barely any oscillations is visible in the RV-method compared to GFEM.

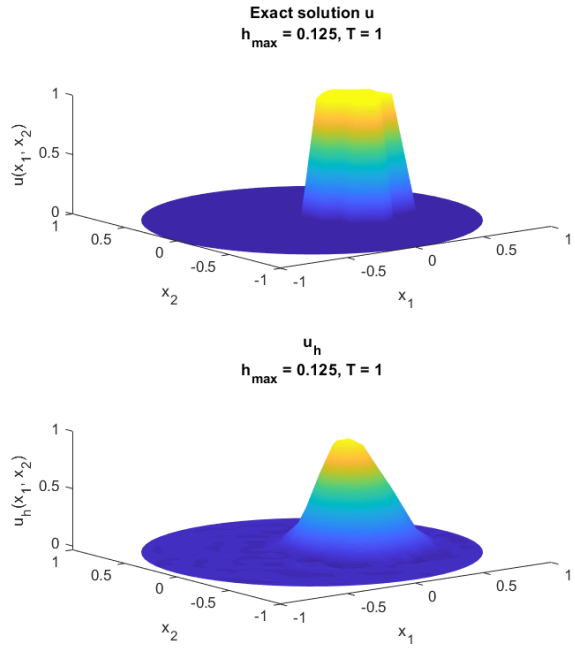

Figure 20: *The results for $h_{max} = \frac{1}{8}$ using the RV-method and step IC. Compared to the analytical solution.*

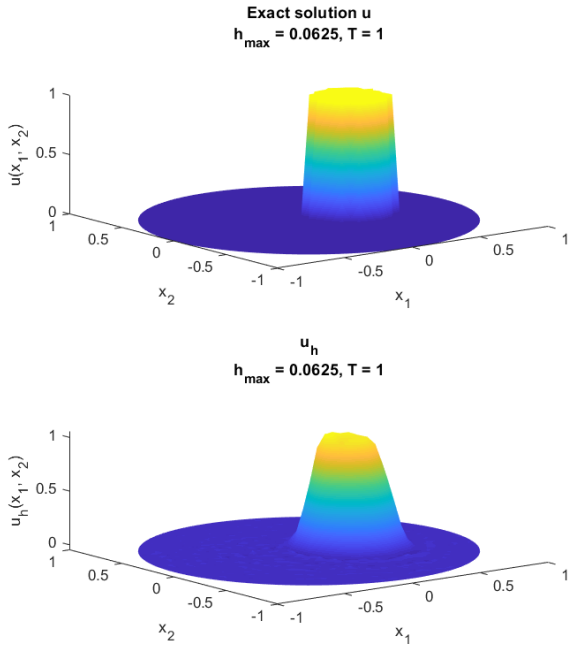

Figure 21: *The results for $h_{max} = \frac{1}{16}$ using the RV-method and step IC. Compared to the analytical solution.*

In Table 4, the calculated L^2-norms and convergence rates q are presented for both the smooth IC and the step IC. It is shown that there is a significant improvement on the convergence for both types of IC:s: the smooth IC averages a rate of $P = 1.97$ (compared to $P = 1.67$ for GFEM) and the step IC has a convergence rate of $P = 0.481$ (compared to $P = 0.223$ for GFEM). The errors and convergence rates are plotted in Figure 22-23. When looking at q for the step IC, it looks like that the convergence rate declines with an increased density of the mesh, however, this effect is probably most visible when using these great values of h_{max}.

Table 4: *The calculated error L2E for each h_{max} and the computed convergence q. L2E is calculated as the L^2-norm, and q as the interpolated slope of L2E between each h_{max}. P is the average convergence rate.*

	Smooth IC		*Step IC*	
$\mathbf{h}_{max}$	**L2E**	**q**	**L2E**	**q**
1/4	0.1877	-	0.2486	-
1/8	5.087E(-2)	1.8837	0.1547	0.6840
1/16	1.277E(-2)	1.9939	0.1156	0.4211
1/32	3.129E(-3)	2.0293	0.09027	0.3564
P	1.9714		0.48057	

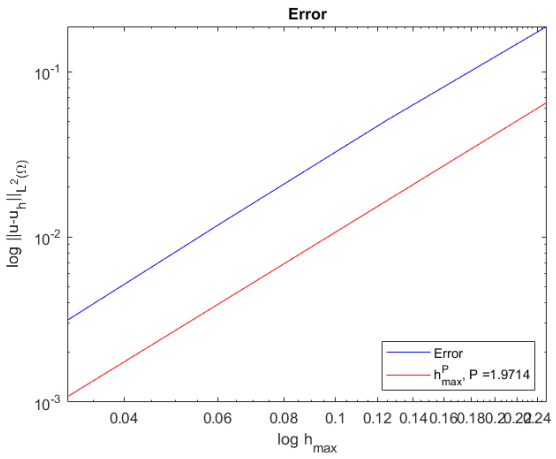

Figure 22: *The calculated $L^2(\Omega)$-norm of the error for Gaussian IC, using the RV-method. Here compared to the average convergence rate.*

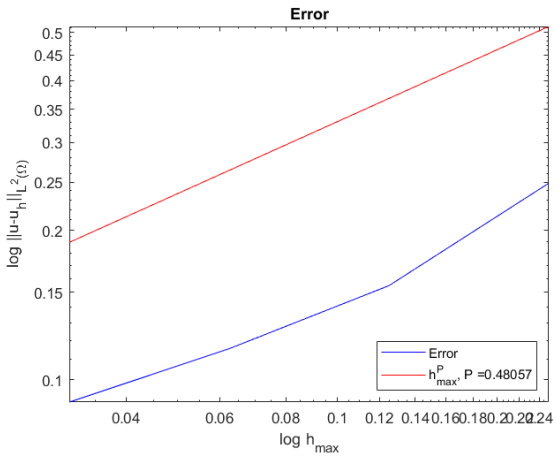

Figure 23: *The calculated $L^2(\Omega)$-norm of the error for step IC, using the RV-method. Here compared to the average convergence rate.*

3.6 Discussion

Both the GLS method and the RV method had dissipating effects on the numerical solutions. When comparing to GFEM, it can be seen that the oscillations was reduced for both methods, however, the convergence rate differed between the methods. GLS reached a convergence rate $P = 1.83$ and $P = 0.302$ for smooth resp. step IC:s. The RV-method reached convergence $P = 1.97$ and $P = 0.48$, which is a significant improvement compared to GFEM for both types of IC:s, as well as a bit better than the GLS method. However, the RV-method had a very long execution time, which is worth to take into consideration when choosing which method to apply.

4 Conclusion

In this project it was demonstrated that it is hard to handle a discontinuous function using FEM, and that it either appears oscillations due to the central difference scheme, or dissipation due to the artificial viscosity that is being introduced. However, when comparing GFEM and the RV-method, it could be seen that the RV-method performed better for both types of IC:s, and with advantage be used for especially discontinuous functions. However, since the the residual needs to be calculated in each step and assembling the R_V matrix in each time step, the execution time is much longer than for the other methods.

For the GLS method, it was shown that it was possible to apply a more stable method that did not take more time than a regular GFEM-solution. There were still some spurious oscillations present, but they were greatly reduced. From this it can therefor be concluded that, depending on the requirements of the project, different stabilization methods can be applied

Appendix A - Code GFEM

```
1
2  %%% Solves the linear advection equation
3  %%% with GFEM using two different initial
4  %%% conditions: one smooth gaussian function
5  %%% and one circular step function.
6  %%%
7  %%% Calls the function mass2D.m and convMat2D.m
8
9  clear all; close all;
10
11 x1_0 = 0.3;
12 x2_0 = 0;
13 r0 = 0.25;
14 CFL = 0.5;
15 hmax = [1/4 1/8 1/16 1/32];
16 T = 1;
17
18 % Choose initial condition
19 n = input('\nType \n 1 for Problem 1.1 \n 2 For Problem 1.3:   ');
20 switch n
21     case 1
22         u0 = @(x1,x2) 1/2.*(1-tanh(((x1-x1_0).^2+(x2-x2_0).^2)./(r0^2)-1));
23         directory = "./Part_1_1/";
24         mkdir(directory);
25     case 2
26         u0 = @(x1,x2) ((x1-x1_0).^2+(x2-x2_0).^2)≤r0^2 *1;
27         directory = "./Part_1_3/";
28         mkdir(directory);
29 end
30
31 for i=1:length(hmax)
32     % sets geometry
33     geometry = @circleg;
34     [p,e,t] = initmesh(geometry, 'hmax', hmax(i));
35
36     % sets values
37     x1 = p(1,:);
38     x2 = p(2,:);
39     u_initial = double(u0(x1,x2))';
40     convTerm = 2*pi*[-x2; x1];
41
42     % Assemble matrices
43     M = mass2D(p,t);
44     C = convMat2D(p, t, convTerm(1,:), convTerm(2,:));
45
46     % Stable time step
47     kn = CFL*hmax(i)/max(sqrt(convTerm(1,:).^2 + convTerm(2,:).^2));
48     kn = T/ceil(T/kn);
49
50     % Crank Nicolson
51     nom = 2*M-kn*C;
52     denom = 2*M+kn*C;
53     A = denom\nom;
54
55     % Sets boundary condition
56     I = eye(length(p));
57     A(e(1 ,:) ,:) = I(e(1 ,:) ,:);
58
59     uh = u_initial;
60     %pdesurf(p,t,uh);
61
```

```
62      time = 0;
63      while (time ≤ T)
64        %set boundary strongly
65        uh(e(1,:)) = 0;
66
67        uh = A*uh;
68
69        %pdesurf(p,t,uh);
70        %drawnow;
71
72        time = time+kn;
73      end
74
75      figure(i);
76      subplot(2,1,1)
77      pdesurf(p,t,u_initial);
78      title(["Exact solution u", "h_{max} = " + hmax(i) + ", T = " + T]);
79      xlabel("x_1");
80      ylabel("x_2");
81      zlabel("u(x_1, x_2)");
82
83      subplot(2,1,2)
84      a = pdesurf(p,t,uh);
85      %%a.EdgeColor = [0 0 0];
86      title(["u_h", "h_{max} = " + hmax(i) + ", T = " + T]);
87      xlabel("x_1");
88      ylabel("x_2");
89      zlabel("u_h(x_1, x_2)");
90
91      colormap('default');
92
93      error = u_initial-uh;
94      L2E(i) = sqrt(error'*M*error);
95
96      % Save the figures
97      filename = directory +"plot_h_" + num2str(hmax(i)) + ".png";
98      saveas(gcf, filename);
99  end
100
101
102 P = polyfit(log10(hmax), log10(L2E),1);
103 figure(i+1)
104 loglog(hmax,L2E,'-b', hmax, hmax.^(P(1)), '-r');
105 title("Error");
106 xlabel("log h_{max}")
107 ylabel("log ||u-u_h||_{L^2(\Omega)}");
108 legend("Error", "h_{max}^P, P =" + P(1),'Location', 'southeast');
109 saveas(gcf, directory+'Error.png');
110
111 %Convergence between each hmax
112 q = zeros(1,length(hmax)-1);
113 for i=1:length(q)
114     q(i) = (log10(L2E(i+1))-log10(L2E(i)))/(log10(hmax(i+1))-log10(hmax(i)));
115 end
```

Appendix B - Code for GLS

```
%%% FEM Part 2 - Stabilization with Galerkin Least Squares

clear all; close all;

x1_0 = 0.3;
x2_0 = 0;
r0 = 0.25;
CFL = 0.5;
hmax = [1/4 1/8 1/16 1/32];
T = 1;

% Choose stuff
n = input('\nType \n 1 for smooth IC \n 2 for step IC:   ');
switch n
    case 1
        u0 = @(x1,x2) 1/2.*(1-tanh(((x1-x1_0).^2+(x2-x2_0).^2)./(r0^2)-1));

    case 2
        u0 = @(x1,x2) ((x1-x1_0).^2+(x2-x2_0).^2)≤r0^2 *1;
end

for i=1:length(hmax)
    geometry = @circleg;
    [p,e,t] = initmesh(geometry, 'hmax', hmax(i));
    x1 = p(1,:);
    x2 = p(2,:);
    u_initial = double(u0(x1,x2))';
    convTerm = 2*pi*[-x2; x1];
    Δ = (1/2)*hmax(i)/max(sqrt(convTerm(1,:).^2 + convTerm(2,:).^2));


    % Assemble matrices
    M = mass2D(p,t);
    C = convMat2D(p, t, convTerm(1,:), convTerm(2,:));
    Sd = SDMat2D(p, t, convTerm(1,:), convTerm(2,:));

    % Stable time step
    kn = CFL*hmax(i)/max(sqrt(convTerm(1,:).^2 + convTerm(2,:).^2));
    kn = T/ceil(T/kn);

    % Crank-Nicolson:
    nom = 1/kn*(M+Δ*C') - 1/2*(C+Δ*Sd);
    denom = 1/kn*(M+Δ*C') + 1/2*(C+Δ*Sd);

    A = denom\nom;

    % Sets boundary condition
    I = eye(length(p));
    A(e(1 ,:) ,:) = I(e(1 ,:) ,:);

    uh = u_initial;
    %pdesurf(p,t,uh);

    time = 0;
    while (time ≤ T)
        %set boundary strongly
        uh(e(1,:)) = 0;
        uh = A*uh;
        % Draws every update
        %pdesurf(p,t,uh);
        %drawnow;
```

```
62
63          time = time+kn;
64      end
65
66      figure(i);
67      subplot(2,1,1)
68      pdesurf(p,t,u_initial);
69      title(["Exact solution u", "h_{max} = " + hmax(i) + ", T = " + T]);
70      xlabel("x_1");
71      ylabel("x_2");
72      zlabel("u(x_1, x_2)");
73
74      subplot(2,1,2)
75      a = pdesurf(p,t,uh);
76      %%a.EdgeColor = [0 0 0];
77      title(["u_h", "h_{max} = " + hmax(i) + ", T = " + T]);
78      xlabel("x_1");
79      ylabel("x_2");
80      zlabel("u_h(x_1, x_2)");
81
82      colormap('default');
83
84      error = u_initial-uh;
85      L2E(i) = sqrt(error'*M*error);
86  end
87
88  P = polyfit(log10(hmax), log10(L2E),1);
89  figure(i+1)
90  loglog(hmax,L2E,'-b', hmax, hmax.^(P(1)), '-r');
91  title("Error");
92  xlabel("log h_{max}")
93  ylabel("log ||u-u_h||_{L^2(\Omega)}");
94  legend("Error", "h_{max}^P, P =" + P(1),'Location', 'southeast');
95
96  %Convergence between each hmax
97  q = zeros(1,length(hmax)-1);
98  for i=1:length(q)
99      q(i) = (log10(L2E(i+1))-log10(L2E(i)))/(log10(hmax(i+1))-log10(hmax(i)));
100 end
```

Appendix C - Code for RV-method

```
%%% GFEM with RV-method
clear all; close all;

x1_0 = 0.3;
x2_0 = 0;
r0 = 0.25;
CFL = 0.5;
C_vel = 0.25;
C_RV = 1;
fprime = @(x1,x2) 2*pi*[-x2; x1];
hmax = [1/4 1/8 1/16 1/32];
T = 1;

% Choose type of IC
n = input('\nType \n 1 for Problem 1.1 \n 2 For Problem 1.3:   ');
switch n
    case 1
        u0 = @(x1,x2) 1/2.*(1-tanh(((x1-x1_0).^2+(x2-x2_0).^2)./(r0^2)-1));
    case 2
        u0 = @(x1,x2) ((x1-x1_0).^2+(x2-x2_0).^2)≤r0^2 *1;
end

for i=1:length(hmax)
    % Setting domain
    geometry = @circleg;
    [p,e,t] = initmesh(geometry, 'hmax', hmax(i));
    x1 = p(1,:);
    x2 = p(2,:);
    numElem = size(t,2);

    u_initial = double(u0(x1,x2))';
    convTerm = 2*pi*[-x2; x1];

    % Stable time step
    kn = CFL*hmax(i)/max(sqrt(convTerm(1,:).^2 + convTerm(2,:).^2));
    kn = T/ceil(T/kn);

    % Assemble matrices
    M = mass2D(p,t);
    C = convMat2D(p, t, convTerm(1,:), convTerm(2,:));

    crankNic = @(Rv) (2*M+kn*C+kn*Rv) \ (2*M-kn*C-kn*Rv);
    I = speye(length(p));

    % Timestepping
    time = 0;
    xi = u_initial;
    xi_prev = xi;
    while time ≤ T

        % calculating and normalizing residual
        Ruh = M\(1/kn*( M*xi-M*xi_prev) + C*(xi));
        Ruh = Ruh/max((xi-mean(xi)));

        % calculating the artificial viscosity based on the residual
        for K = 1:numElem
            nodes = t(1:3,K);
            coords = p(:,nodes);
            d = [norm(coords(:,1)-coords(:,2)) norm(coords(:,2)-coords(:,3))...
                norm(coords(:,3)-coords(:,1))];
            h_K = min(d);
```

```
62
63              fp = fprime(coords(1,:), coords(2,:));
64
65              beta_K = max(sqrt(fp(1,:).^2+fp(2,:).^2));
66              Res_K = max(abs(Ruh(nodes)));
67              eps(K) = min(C_vel*h_K*beta_K, C_RV*h_K^2*Res_K);
68          end
69
70          % Assembling matrices
71          Rv = RvMat2D(p,t,eps);
72          A = sparse(crankNic(Rv));
73          A(e(1 ,:) ,:) = I(e(1 ,:) ,:);
74
75          xi_prev = xi;
76
77          xi = A*xi_prev;
78          xi(e(1,:)) = 0;
79          % Draws every update
80          %pdesurf(p,t,xi);
81          %drawnow;
82          time = time+kn;
83      end
84
85      figure(i);
86      subplot(2,1,1)
87      pdesurf(p,t,u_initial);
88      title(["Exact solution u", "h_{max} = " + hmax(i) + ", T = " + T]);
89      xlabel("x_1");
90      ylabel("x_2");
91      zlabel("u(x_1, x_2)");
92
93      subplot(2,1,2)
94      pdesurf(p,t,xi);
95      title(["u_h", "h_{max} = " + hmax(i) + ", T = " + T]);
96      xlabel("x_1");
97      ylabel("x_2");
98      zlabel("u_h(x_1, x_2)");
99
100     colormap('default');
101
102     error = u_initial-xi;
103     L2E(i) = sqrt(error'*M*error);
104 end
105
106 P = polyfit(log10(hmax), log10(L2E),1);
107 figure(i+1)
108 loglog(hmax,L2E,'-b', hmax, hmax.^(P(1)), '-r');
109 title("Error");
110 xlabel("log h_{max}")
111 ylabel("log ||u-u_h||_{L^2(\Omega)}");
112 legend("Error", "h_{max}^P, P =" + P(1),'Location', 'southeast');
113
114 %Convergence between each hmax
115 q = zeros(1,length(hmax)-1);
116 for i=1:length(q)
117     q(i) = (log10(L2E(i+1))-log10(L2E(i)))/(log10(hmax(i+1))-log10(hmax(i)));
118 end
```

Appendix D - Code for Assembling Matrices

```
1
2  %%% Copied from the book
3  %%% "The Finite Element Method: Theory, Implementation, and
4  %%% Practice": Larson, Bengtzon
5  %%%
6  %%% Creates mass matrix for 2D-problems.
7
8  function M = mass2D(p,t)
9  N = size(p,2);
10 M = sparse(N,N);
11 %M = zeros(N,N);
12 for K =1:size(t,2)
13     nodes = t(1:3,K);
14     x = p(1,nodes);
15     y = p(2,nodes);
16     area_K = polyarea(x,y);
17
18     Mk = (1/12)*[2 1 1; 1 2 1; 1 1 2]*area_K;
19     M(nodes, nodes) = M(nodes,nodes) + Mk;
20 end
```

```
1
2  %%% Copied from the book
3  %%% "The Finite Element Method: Theory, Implementation, and
4  %%% Practice": Larson, Bengtzon
5  %%%
6  %%% Creates convection matrix for 2D-problems.
7
8  function C = convMat2D(p,t,bx,by)
9  np=size(p,2);
10 nt=size(t,2);
11 %C=sparse(np,np);
12 C = zeros(np,np);
13 for i=1:nt
14     loc2glb=t(1:3,i);
15     x=p(1,loc2glb);
16     y=p(2,loc2glb);
17     [area,b,c]=Gradients(x,y);
18
19     bxmid=mean(bx(loc2glb));
20     bymid=mean(by(loc2glb));
21     CK=ones(3,1)*(bxmid*b+bymid*c)'*area/3;
22     C(loc2glb,loc2glb)=C(loc2glb,loc2glb)+CK;
23 end
24 end
```

```
1  %%% Copied from the book
2  %%% "The Finite Element Method: Theory, Implementation, and
3  %%% Practice%: Larson, Bengtzon
4  %%% with a slight modificaiton to introduce epsilon as stabilization.
5  %%%
6  %%% Constructs stabilization for 2D-problems.
7  function Rv = RvMat2D(p,t, eps)
8  N = size(p,2);
9  Rv = sparse(N,N);
10 %Rv = zeros(N,N);
11
```

```
for K =1:size(t,2)
    nodes = t(1:3,K);
    x = p(1,nodes);
    y = p(2,nodes);
    area_K = polyarea(x,y);
    bi = [y(2)-y(3); y(3)-y(1); y(1)-y(2)]/(2*area_K);
    ci = [x(3)-x(2); x(1)-x(3); x(2)-x(1)]/(2*area_K);
    Rv_K = eps(K)*(bi*bi'+ci*ci')*area_K;
    Rv(nodes,nodes) = Rv(nodes,nodes)+Rv_K;
end
end
```

```
%%% Copied from the book
%%% "The Finite Element Method: Theory, Implementation, and
%%% Practice%: Larson, Bengtzon
%%%
%%% Creates streamline diffusion matrix for 2D-problems.

function Sd = SDMat2D(p,t,bx,by)
np=size(p,2);
nt=size(t,2);
Sd=sparse(np,np);
for i=1:nt
loc2glb=t(1:3,i);
x=p(1,loc2glb);
y=p(2,loc2glb);
[area,b,c]=Gradients(x,y);
bxmid=mean(bx(loc2glb));
bymid=mean(by(loc2glb));
SdK=(bxmid*b+bymid*c)*(bxmid*b+bymid*c)'*area;
Sd(loc2glb,loc2glb)=Sd(loc2glb,loc2glb)+SdK;
end
```