Uppsala University

Advanced Numerical Methods

1TD050

Assignment 1

Author:
Csongor Horváth

October 9, 2023

Introduction

In this report we are investigating the numerical solution of the following given PDE problem:

Let Ω be a fixed (open) domain in $\mathbb{R}^2$, with boundary $\partial\Omega$ over a time interval $[0, T]$ with initial time zero and the final time T. We are interested in solving the following time-dependent scalar conservation laws:

$$\begin{aligned} \partial_t u + \nabla \cdot \boldsymbol{f}(u) &= 0, \qquad && (\boldsymbol{x}, t) \in \Omega \times (0, T], \\ u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), && \boldsymbol{x} \in \Omega, \end{aligned}$$

with appropriate boundary conditions. Here u represents the unknown variable, $f \in \mathcal{C}^1(\mathbb{R}, \mathbb{R}^2)$ is the flux term and $u_0 \in L^\infty(\mathbb{R}^2)$ is given initial data. Let $0 = t_0 < t_1 < \ldots < t_N = T$ be a sequence of discrete time steps with associated time intervals $I_n = (t_{n-1}, t_n]$ of length $k_n = t_n - t_{n-1}, n = 1, 2, \cdots, N$. Let

$$\mathcal{X}_h := \left\{ v : v \in H^1(\Omega), v(\boldsymbol{x}) - \text{ cont. pw. linear in } \Omega \right\},$$

be a finite element space consisting of continuous piecewise linear polynomials on a mesh $\mathcal{T}_h = \{K\}$ of mesh-size $h(\boldsymbol{x})$ and let $V_{h,0}$ be the space of all functions in $\mathcal{X}_h$ vanishing on $\partial\Omega$. Next, let us denote by $U_0 = \hat{\pi}_h u_0$ the interpolation of the initial data into the finite element space $\mathcal{X}_h$. We are now ready to formulate the following finite element approximation: for $n = 1, 2, \cdots, N$ find $U_n \in \mathcal{X}_h$ such that

$$\frac{1}{k_n}\left(U_n - U_{n-1}, v\right) + \frac{1}{2}\left(\nabla \cdot \left(\boldsymbol{f}\left(U_{n-1}\right) + \boldsymbol{f}\left(U_n\right)\right), v\right) = 0, \quad \forall v \in \mathcal{X}_h, \qquad (1)$$

where $U_n = U_h\left(t_n\right)$ is the solution at the discrete time steps t_n. The implicit Crank-Nicholson time-stepping is used for the time discretization.

Note: In the tasks we will use $\Omega = \{\mathbf{x} : x_1^2 + x_2^2 \leq 1\}$, and Dirichlet boundary condition.

Part I

Problem 1.1

In this task we should implement a Matlab code to solve the finite element approximation given in (1). The implementation should be done with the following values: $u_0(\mathbf{x},0) = \frac{1}{2}\left(1 - \tanh\left(\frac{(x_1-x_1^0)^2+(x_2-x_2^0)^2}{r_0^2} - 1\right)\right)$ is the initial data with $r_0 = 0.25$, $(x_1^0, x_2^0) = (0.3, 0)$. Additionally we should use $T = 1$, $CFL = 0.5$ and create a solution using $h_{max} = \frac{1}{8}$ and $h_{max} = \frac{1}{16}$ mesh size.

First let's rewrite the flux term of equation (1) in the given way. So use $\nabla \cdot \boldsymbol{f}(u) := \boldsymbol{f'}(u) \cdot \nabla u$, assuming $\boldsymbol{f'}(u) = 2\pi(-x_2, x_1)$. Then knowing that ∇ is a linear operator, we get this:

$$\frac{1}{k_n}(U_n - U_{n-1}, v) + \frac{1}{2}(\boldsymbol{f'}(U_{n-1}) \cdot \nabla U_{n-1} + \boldsymbol{f'}(U_n) \cdot \nabla U_n, v) = 0, \quad \forall v \in \mathcal{X}_h,$$

After this step, we should create the system of linear equations from this FEM formulation using the appropriate basis functions. Then we can solve the equation system for U_{n+1} in case U_n is given using the given u_0 value.

We can write U_n in the form $U_n = \sum \xi_i \varphi_i$, where the sum is taken from the φ_i basis function of $\mathcal{X}_h$ space. And in the form of the basis function the statement that $\forall v \in \mathcal{X}_h$, is equivalent with the statement that the equation holds for all basis function. From this we get the following for of the equation:

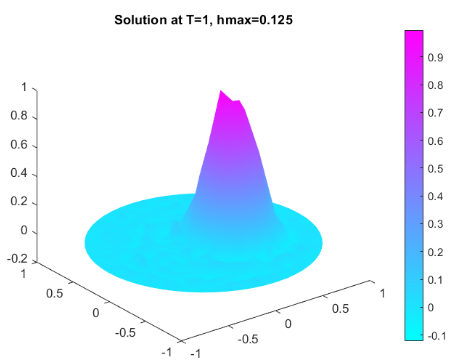
$$\frac{1}{k_n}\sum_{N_i \in \mathcal{N}_h}(\xi_{i,n} - \xi_{i,n-1}) \cdot \int_\Omega \varphi_j \varphi_i d\boldsymbol{x} + \frac{1}{2}\sum_{N_i \in \mathcal{N}_h}(\xi_{i,n} + \xi_{i,n-1}) \cdot \int_\Omega \boldsymbol{f'}(u) \nabla \varphi_j \varphi_i d\boldsymbol{x} = 0 \quad \forall i$$

Now we can establish the M mass matrix from the values of $\int_\Omega \varphi_j \varphi_i$ and C convection matrix with values from $\int_\Omega \boldsymbol{f'}(u) \nabla \varphi_j \varphi_i$. Using these matrices we can write the equation of FEM with implicit Crank-Nicholson time-discretization in the following form, which solution can be implemented easily:

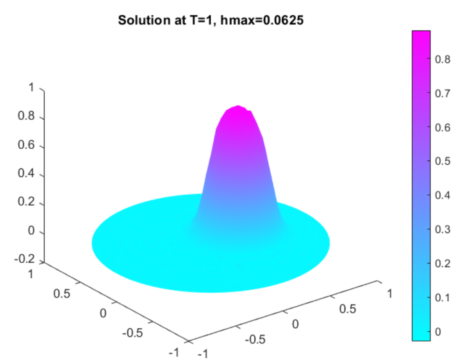
$$\frac{1}{k_n}\boldsymbol{M}(\boldsymbol{\xi_n} - \boldsymbol{\xi_{n-1}}) + \frac{1}{2}\boldsymbol{C}(\boldsymbol{\xi_n} + \boldsymbol{\xi_{n-1}}) = 0$$

Notes on the implementation: For the calculation of mass and convection matrix I follow the instructions given in the book *The Finite Element Method: Theory, Implementation, and Applications*

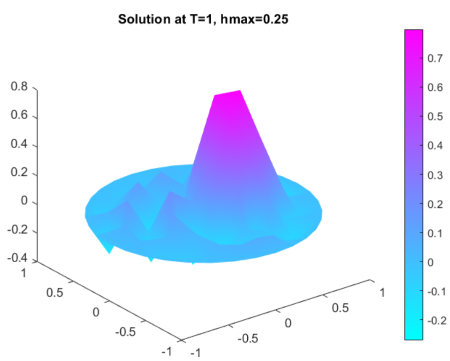
The plots of the final solutions at $T = 1$ with $hmax \in \{\frac{1}{8}, \frac{1}{16}\}$ can be found in Figure 1. As it can be seen the problem is a conservation problem, so the solution if the initial data in the given case. It can be seen that solution contains some noise for both meshes over time for both mesh. This means that even if we start from the exact solution, over time some noise occurs in the numerical solution of the equation even with finer meshes. Now let's move to the next part, where we investigate the size of the generated error.



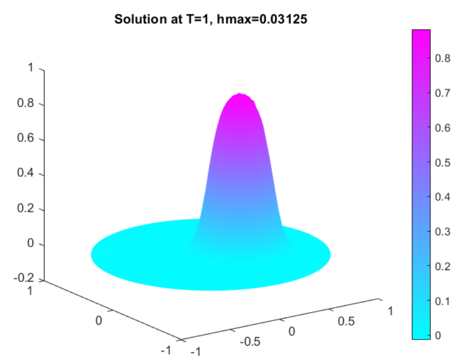
(a) Solution for $hmax = 1/8$



(b) Solution for $hmax = 1/16$



(c) Solution for $hmax = 1/4$



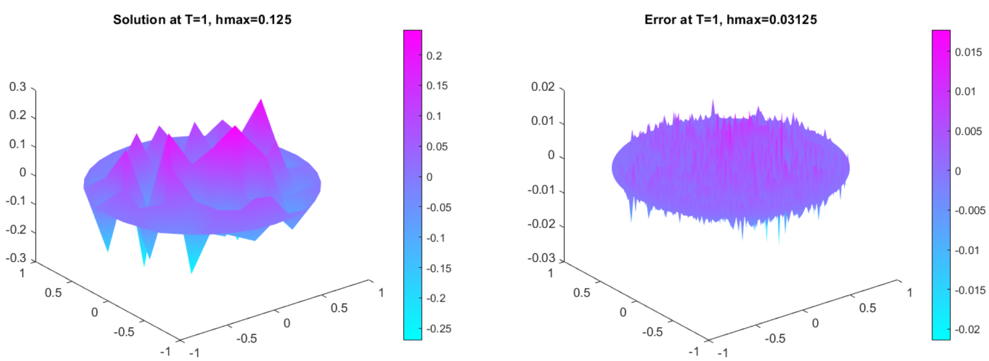
(d) Solution for $hmax = 1/32$

Figure 1: FEM solutions of the PDE with smooth initial data

Problem 1.2

For calculating the error, we need to know the exact solution. As I mentioned in this case that is the initial data. So the error can be calculated $e = u - U_n = u_0 - U_n \approx U_0 - U_n =: e_n$. As it is given in the assignment the norm of the error can be calculated in the following way in the nth time step: $||e||_{L^2(\Omega)} = \left(\int_\Omega e^2 dx\right)^{\frac{1}{2}} \approx \sqrt{e_n^T M e_n}$, where M is the mass matrix.

Now we should add this to the Matlab implementation and run the simulation for $hmax \in \{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$ and calculate the α convergence rate. The errors of the solution can be seen in Figure 2 for the smallest and largest mesh size. As it can be seen the error is getting less with using smaller mesh size.



(a) Solution for $hmax = 1/4$

(b) Error for $hmax = 1/32$

Figure 2: FEM, error of the PDE at $T = 1$, smooth initial data

The $\alpha = 1.18$ convergence rate parameter is calculated based on the error norms and a linear interpolation of the log values. The log-log plot of the errors can be seen in Figure 3 together with the h_{max} versus h_{max}^α plot.

Actually with sufficient parameter it should be possible to reach second order convergence for the FEM algorithm with the smooth initial data. To investigate this I modified the time stepping parameter CFL and changed it to 0.1 from 0.5. After this I got a much better convergence rate with $\alpha = 1.74$, which can be seen in Figure 4. This is much closer to the expected second order convergence.
(*Note: with the same change there wasn't any significant improvement on the*

α convergence rate with the non-smooth initial data).

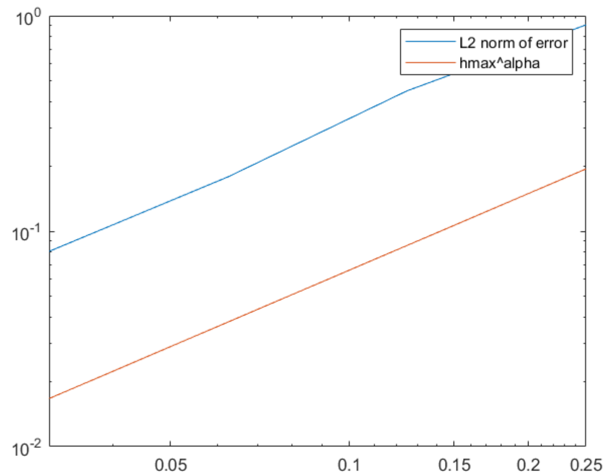


Figure 3: FEM $\alpha = 1.18$ convergence rate with smooth initial data

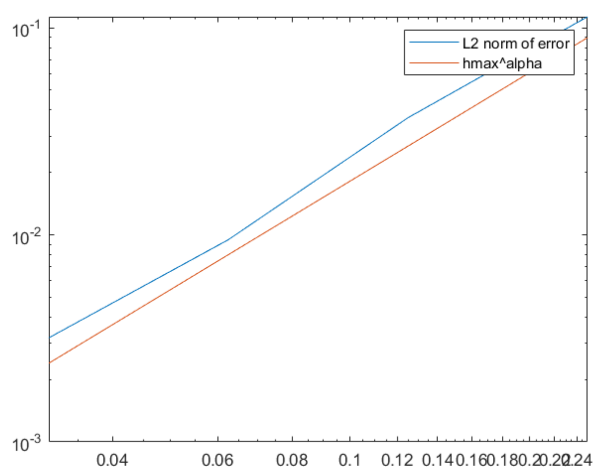


Figure 4: FEM $\alpha = 1.74$ convergence rate, smooth initial data, $CFL = 0.1$

Problem 1.3

In this part the task was to solve the same equation as in the previous two parts with different initial data.

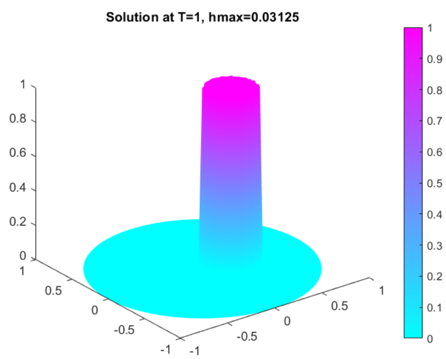


Figure 5: The non-smooth initial data

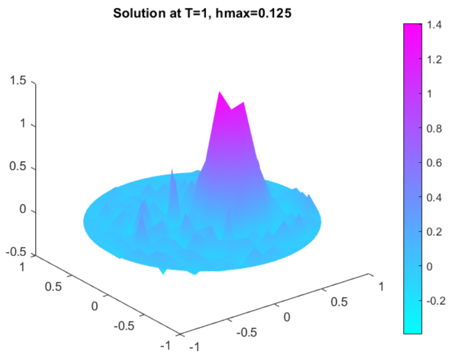
The theoretical formulation of the problems work the same way with different u_0 data. So the only thing to complete this task was to change the initial data in the MATLAB code and create similar figures as before.

Now the initial data is set to:

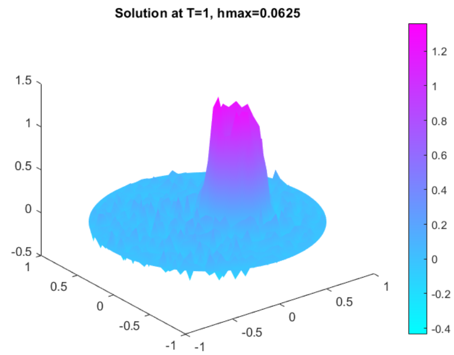
$$u_0(\boldsymbol{x}, 0) = \begin{cases} 1 & \text{if } (x_1 - x_1^0)^2 + (x_2 - x_2^0)^2 \leq r_0^2 \\ 0 & \text{otherwise} \end{cases}$$

So now the initial data is non-smooth and therefore the simulation is expected to result in worse results. As for a reference the initial data is plotted in Figure 5.

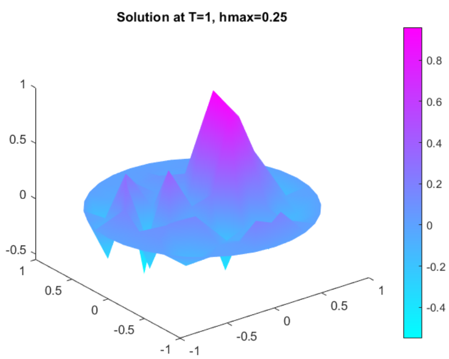
The plots for the solutions at the final time ($T = 1$) with different mesh sizes can be seen in Figure 6. As it could be expected these results are more noisy then the ones in the case of the smooth initial data. But as the mesh is getting finer, the solution is getting better and better. To measure this phenomenon now let's look at the error plots and the convergence rate.
The errors for the smallest and largest mesh is plotted in Figure 7. The shape of the error is similar to the previous case, but the scale is much larger as it



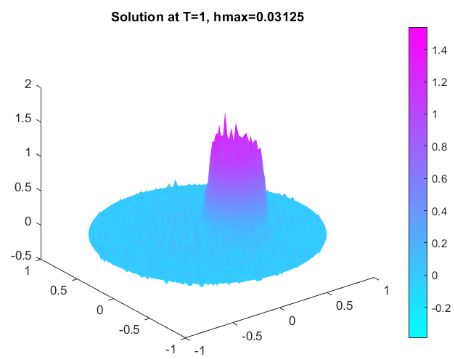
(a) Solution for $hmax = 1/8$



(b) Solution for $hmax = 1/16$



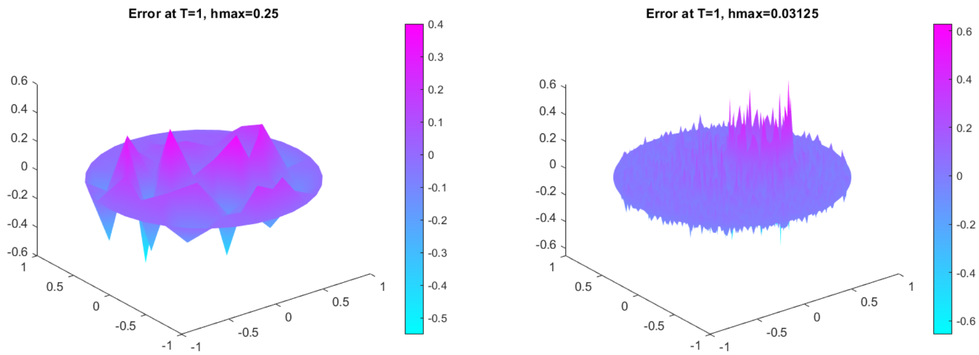
(c) Solution for $hmax = 1/4$



(d) Solution for $hmax = 1/32$

Figure 6: FEM solutions of the PDE, non-smooth initial data

can be expected. Now let's look at the convergence rate plotted in Figure 8. The calculated convergence rate is $\alpha = 0.28$, which is much less, than in the previous case. So as it was expected from the numerical solutions we also get that the convergence rate is much worse in this case as in the smooth version of the problem.



(a) Solution for $hmax = 1/4$ (b) Error for $hmax = 1/32$

Figure 7: FEM, error of the PDE at $T = 1$, non-smooth initial data

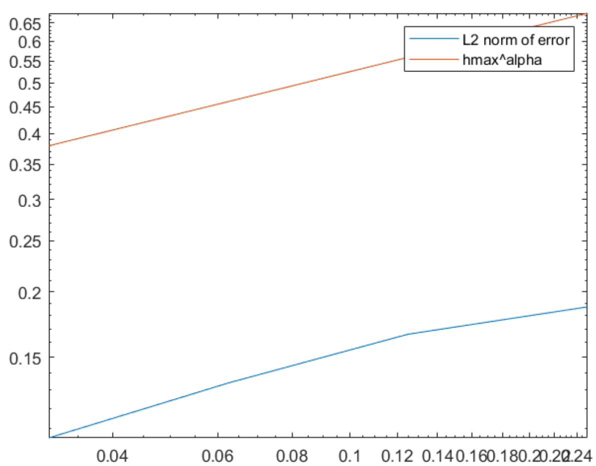


Figure 8: FEM $\alpha = 0.28$ convergence rate for the non-smooth initial data

Part II

In part 2, we will investigate methods for stabilizing the solution for the same problem as the one in part 1. To this end the SUPG and RV method for this problem will be introdued and implemented for both smooth and non smooth initial data.

Problem 2.1

First we have to create the SUPG formulation of the original problem:

$$\begin{aligned} \partial_t u + \nabla \cdot \boldsymbol{f}(u) &= 0, \qquad && (\boldsymbol{x}, t) \in \Omega \times (0, T], \\ u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), && \boldsymbol{x} \in \Omega, \end{aligned}$$

Reformulate the problem with the given information $\nabla \boldsymbol{f}((u)) = \boldsymbol{f'}(u) \cdot \nabla u$.

$$\begin{aligned} \partial_t u + \boldsymbol{f'}(u) \cdot \nabla u &= 0, \qquad && (\boldsymbol{x}, t) \in \Omega \times (0, T], \\ u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), && \boldsymbol{x} \in \Omega, \end{aligned}$$

Now we can get the SUPG formulation of the problem by testing with $v + \delta \boldsymbol{f'}(v) \cdot \nabla v$. Find $u_h \in X_h$ s.t.:

$$(\partial_t u_h + \boldsymbol{f'}(u_h) \cdot \nabla u_h, v) + (\partial_t u_h + \boldsymbol{f'}(u_h) \cdot \nabla u_h, \delta \boldsymbol{f'}(v) \cdot \nabla v) = 0, \quad \forall v \in X_h \tag{2}$$

Note: Here $\boldsymbol{f'}(u_h)$ only depends on the position x, therefore it is a linear PDE, even though in this form $\boldsymbol{f'}(u_h)$ seems to be dependent on the solution. So using the implicit Cranck-Nicholson time-stepping, we get the following formulation:

$$\frac{1}{k}(U_n - U_{n-1}, v + \delta \boldsymbol{f'}(v) \cdot \nabla u) + \frac{1}{2}(\boldsymbol{f'}(U_n) \cdot \nabla U_n + \boldsymbol{f'}(U_{n-1}) \cdot \nabla U_{n-1}, v + \delta \boldsymbol{f'}(v) \cdot \nabla v) = 0$$

To prove stability we will use the form written in (2) and use $v = u$ to make stability estimates.

$$\frac{1}{2}\partial_t \|u_h\|^2 + \|\sqrt{\delta} \boldsymbol{f'}(u_h) \nabla u_h\|^2 + (\boldsymbol{f'}(u_h) \nabla u_h, u_h) + (\partial_t u_h, \delta \boldsymbol{f'}(u_h) \nabla u_h) = 0$$

With the notations of the lectures we get that since $\overline{\beta} = \boldsymbol{f'}$ is divergence free in this case as $\frac{x_2}{\partial_{x_1}} = \frac{x_1}{\partial_{x_2}} = 0$. So $\nabla \cdot \boldsymbol{f'} = 0$. Therefore the same prove works as in the lecture to show that $(\boldsymbol{f'} \cdot \nabla u_h, u_h) = 0$ as it was given in the lecture for $(\overline{\beta} \cdot \nabla u_h, u_h)$ with the homogeneous Dirichlet boundary condition.

For the other term we can show the same thing using the fully discrete form. So in the fully discrete form we get $(\frac{U_n - U_{n-1}}{k_n}, \delta \boldsymbol{f'}(\boldsymbol{x}) \nabla \frac{U_n + U_{n-1}}{2}) = 0$.

Therefore in the fully discrete form we showed stability by gaining $\frac{1}{2}\partial_t \|U_h\|^2 + \|\sqrt{\delta}\boldsymbol{f'}(x)\delta U_h\| = 0$ in the fully discrete equivalent.

Therefore we showed stability.

Problem 2.2 - 2.3

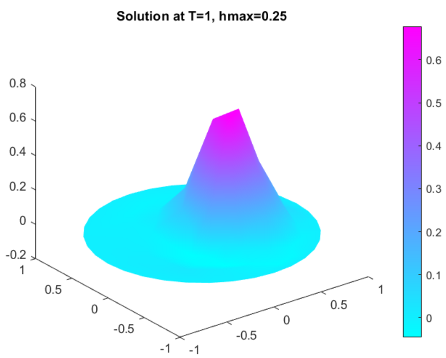
The SUPG (Streamline-Upwind-Petrov-Galerkin) method

First let's discuss and implement the SUPG method. From the the previous discussion we obtain the following discretized system to be implemented:

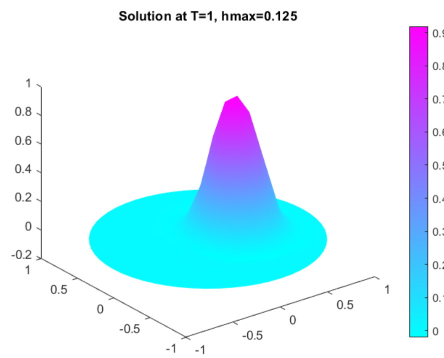
$$\xi_{n+1}\left(\frac{M}{k_n}+\frac{C}{2}+\frac{SM}{k_n}+\frac{SC}{2}\right)=\xi_n\left(\frac{M}{k_n}-\frac{C}{2}+\frac{SM}{k_n}-\frac{SC}{2}\right)$$

Here M and C are the mass and convection matrices from part 1. $SM_{ij} = (\varphi_j, \delta \boldsymbol{f'}(N_i)\nabla\varphi_i)$, and $SC_{ij} = (\boldsymbol{f'}(N_j)\nabla\varphi_j, \delta\boldsymbol{f'}(N_i)\nabla\varphi_i)$. Here $\boldsymbol{f'}(N_i)$ is used to show that the value of $\boldsymbol{f'}$ is only space dependent in this case, and N_i denotes the node of the triangulation corresponding to the φ_i basis functions location. From here the implementation is straightforward using discrete integration to calculating the matrices.

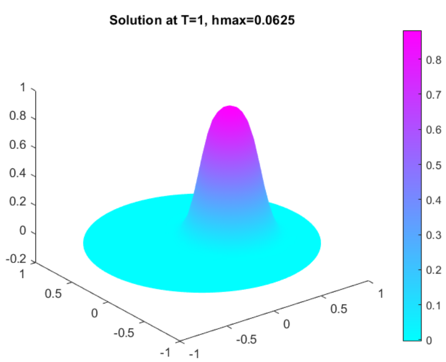
The plots for the smooth initial data can be found in Figures 9, 10 and 11. While the plots for the non-smooth initial data can be found in Figures 12,13, 14. As from the plots and error it can be seen the constant 1 initial data parts stays 0 and there is basically no noise to it only close to the "interesting" part of the function. Also the non-smooth data act much better in this case than with the original FEM. This is not surprising as this method is stable. Also we get higher convergence rates with $\alpha = 1.82$ and $\alpha = 0.29$ for the two cases. This means that the SUPG method is also superior to basic FEM in the rate of convergence.



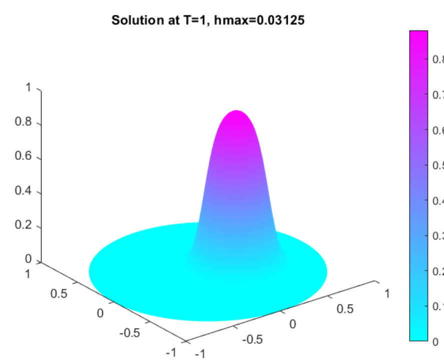
(a) Solution for $hmax = 1/4$



(b) Solution for $hmax = 1/8$

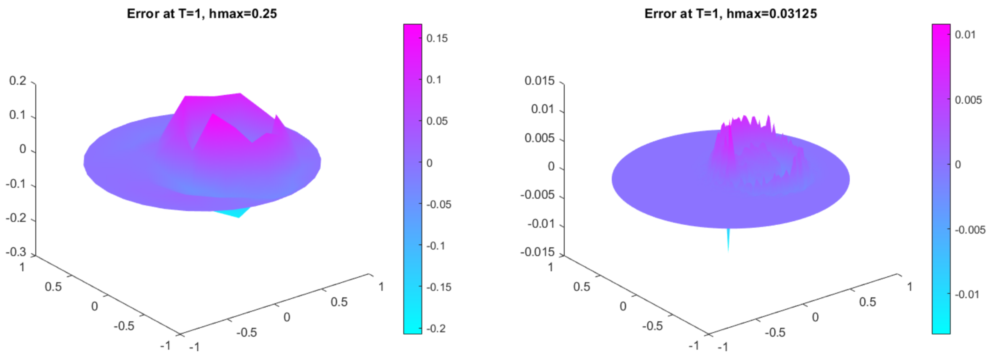


(c) Solution for $hmax = 1/16$



(d) Solution for $hmax = 1/32$

Figure 9: SUPG solutions of the PDE, smooth initial data



(a) Error for $hmax = 1/4$ (b) Error for $hmax = 1/32$

Figure 10: SUPG error of the PDE, smooth initial data

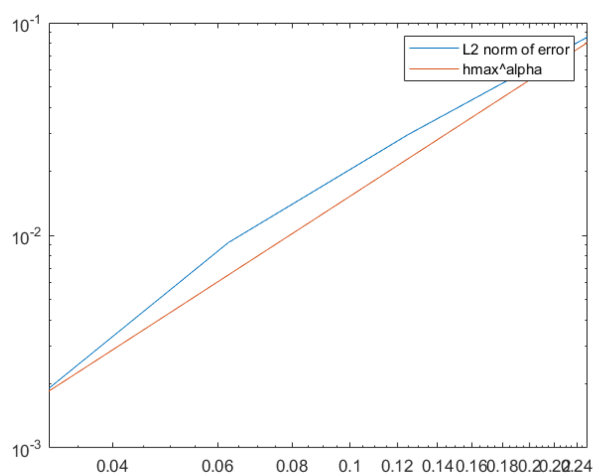
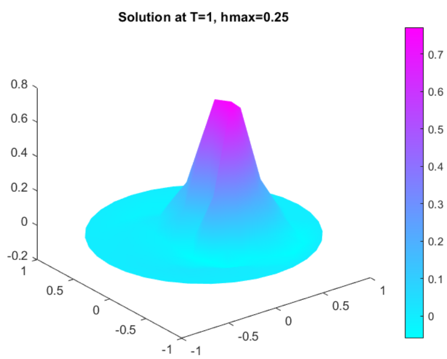
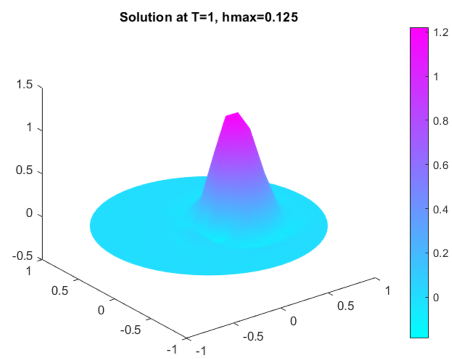


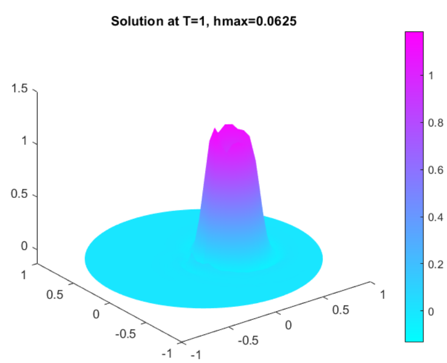
Figure 11: $\alpha = 1.82$ convergence rate with SUPG method, smooth initial data



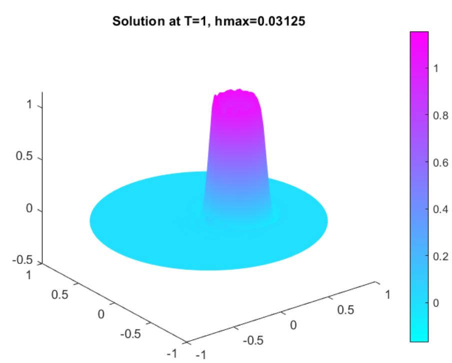
(a) Solution for $hmax = 1/4$



(b) Solution for $hmax = 1/8$

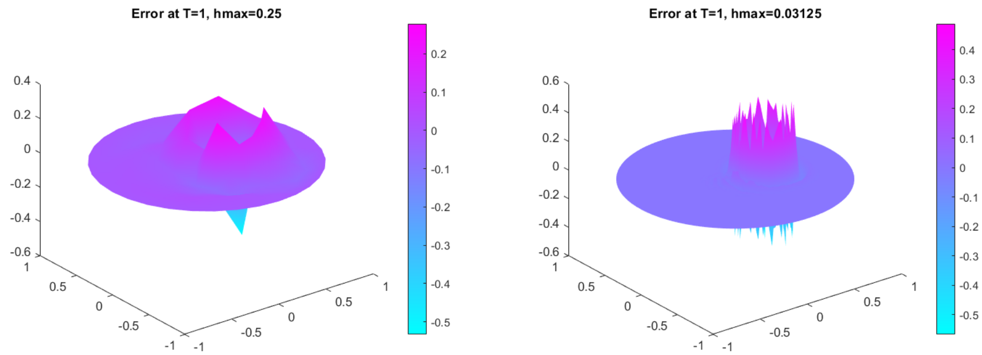


(c) Solution for $hmax = 1/16$



(d) Solution for $hmax = 1/32$

Figure 12: SUPG solutions of the PDE, non-smooth initial data



(a) Error for $hmax = 1/4$ (b) Error for $hmax = 1/32$

Figure 13: SUPG error of the PDE, non-smooth initial data

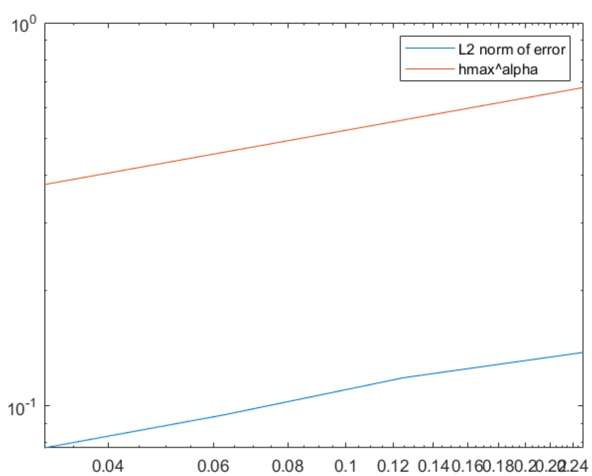


Figure 14: $\alpha = 0.29$ convergence rate with SUPG method, non-smooth initial data

The RV (Residual based artificial viscosity) method

The RV method was given in the task description. In this case we add an additional term to the equation.

$$\frac{1}{k_n}\left(U_n - U_{n-1}, v\right) + \frac{1}{2}\left(\nabla \cdot \left(\boldsymbol{f}\left(U_{n-1}\right) + \boldsymbol{f}\left(U_n\right)\right), v\right) + \frac{1}{2}\left(\varepsilon_n\left(U_{n-1}\right)\nabla\left(U_n + U_{n-1}\right), \nabla v\right) = 0, \quad \forall v \in \mathcal{X}_h,$$

Here the artificial viscosity $\varepsilon_n := \varepsilon_h\left(t_n\right)$ is computed as

$$\varepsilon_{n,K} = \min\left(C_{\mathrm{vel}} h_K \beta_K, C_{\mathrm{RV}} h_K^2 \frac{\|R_{\mathrm{RV}}\|_{\infty,K}}{\left\|U_n - \overline{U_n}\right\|_{\infty,\Omega}}\right),$$

for each cell $K \subset \mathcal{T}_h$. Here $R\left(U_n\right) = \frac{1}{k_n}\left(U_n - U_{n-1}\right) + \nabla \cdot \boldsymbol{f}\left(U_n\right)$, and $\left\|U_n - \overline{U_n}\right\|_{\infty,\Omega}$ is a normalization term, with $\overline{U_n}$ being the space average of the solution over Ω, h_K is the mesh-size of the element K, β_K denotes the local element wave speed that is computed as

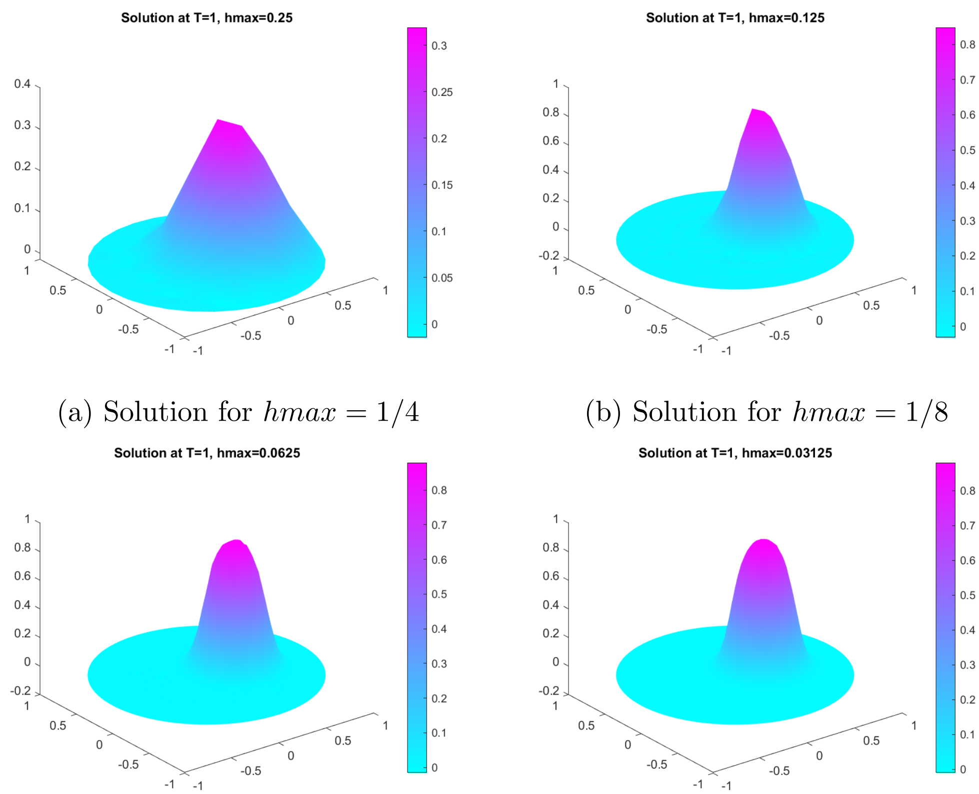
$$\beta_K \equiv \|\boldsymbol{f}'\left(U_n\right)|\|_{L^\infty(K)} = \max_{N_i \in K, i=0,1,2}\left(\left[\left(f_1'\left(U_n\right)\right)^2 + \left(f_2'\left(U_n\right)\right)^2\right]^{\frac{1}{2}}\left(N_i\right)\right)$$

, where $N_i, i = 0, 1, 2$ is the nodes of the element $K, C_{\mathrm{vel}} = 0.25$ and $C_{\mathrm{RV}} = 1$ are stabilization parameters. The standard Galerkin solution can be obtained by setting $C_{\mathrm{vel}} = 0$.

From these given information it is straightforward to do the implementation. All we need is one function to calculate $\epsilon_{n,K}$ and one to calculate the matrix for the new term.

Note: From (R_u, v) the value of R_u can be calculated as $R(U_n) = M^{-1}(M(\frac{U_n - U_{n-1}}{k_n}) + CU_n)$ based on the given form.

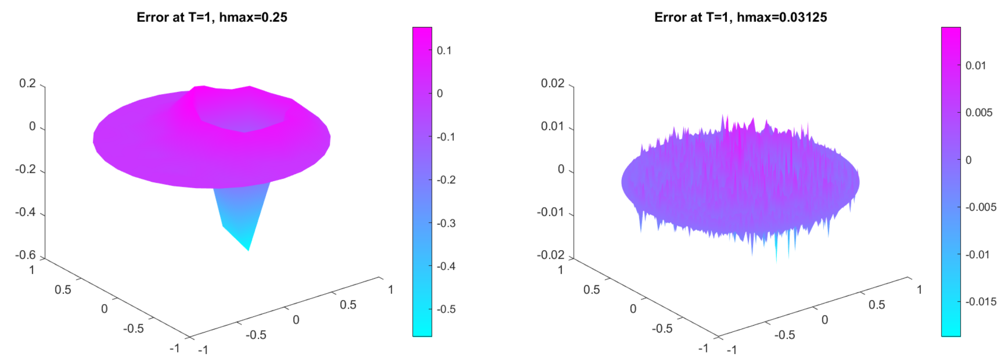
The plots for the smooth initial data can be found in Figures 15, 16 and 18. While the plots for the non-smooth initial data can be found in Figures 19,20, 22. In this case for both the smooth and non-smooth data the plots for small mesh sizes are worse than for FEM and SUPG, but as the mesh get finer, the results start to act nicely. Also the convergence rates are here the bests with $\alpha = 1.97$ and $\alpha = 0.53$. The $\epsilon(u)$ values at the final time is also plotted in Figure 17 and 21. Though this plots are not accurate as ϵ is $\tau_h = \{K\} \mapsto \mathcal{R}$ and the plots are done at the nodal points with calculating an average from the neighbouring elements ϵ value.



(a) Solution for $hmax = 1/4$ (b) Solution for $hmax = 1/8$

(c) Solution for $hmax = 1/16$ (d) Solution for $hmax = 1/32$

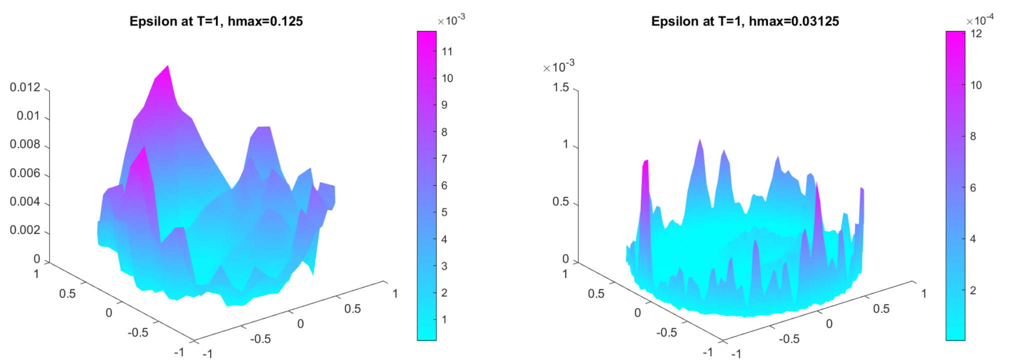
Figure 15: RV solutions of the PDE, smooth initial data



(a) Error for $hmax = 1/4$

(b) Error for $hmax = 1/32$

Figure 16: RV, error of the PDE, smooth initial data



(a) $\epsilon(U_n), T = 1$ for $hmax = 1/4$

(b) $\epsilon(U_n), T = 1$ for $hmax = 1/32$

Figure 17: Value of $\epsilon(U)$ at the end of the RV simulation, smooth initial data

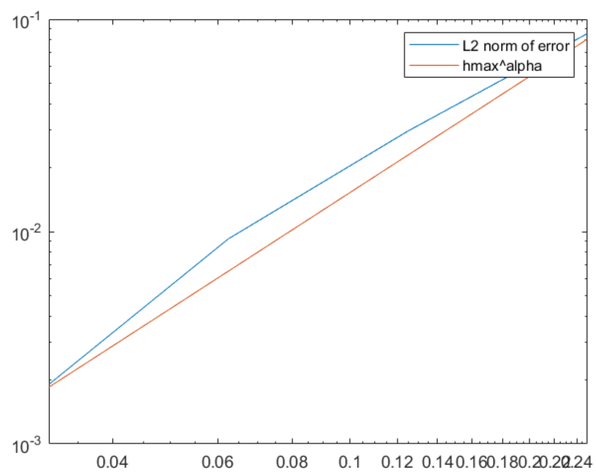
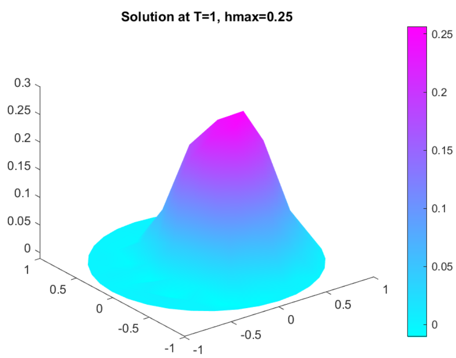
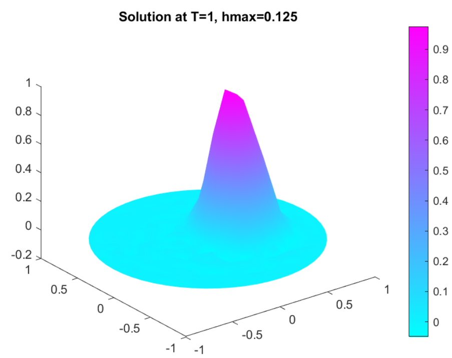


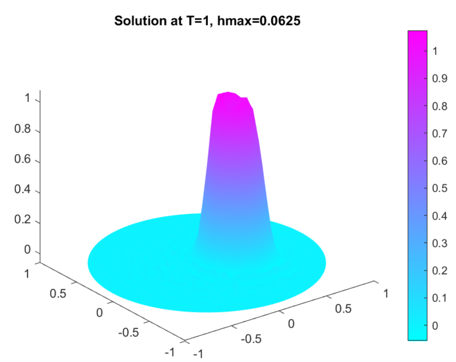
Figure 18: $\alpha = 1.97$ convergence rate with RV method, smooth initial data



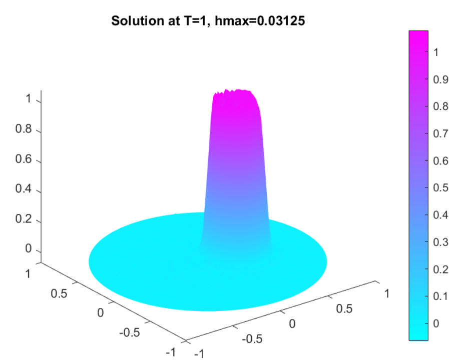
(a) Solution for $hmax = 1/4$



(b) Solution for $hmax = 1/8$

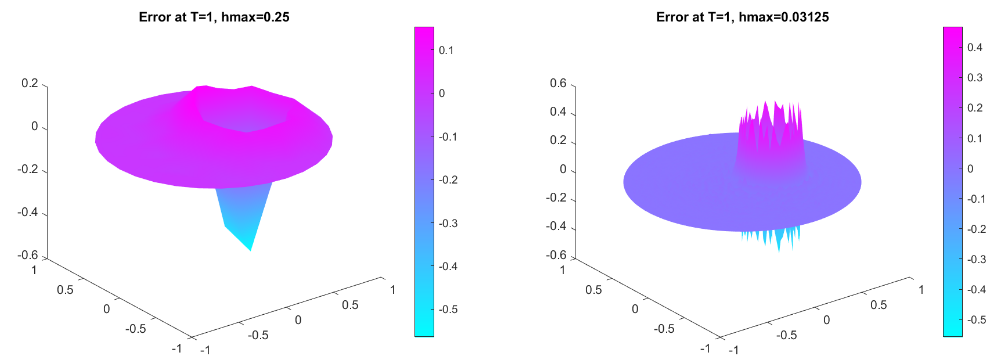


(c) Solution for $hmax = 1/16$



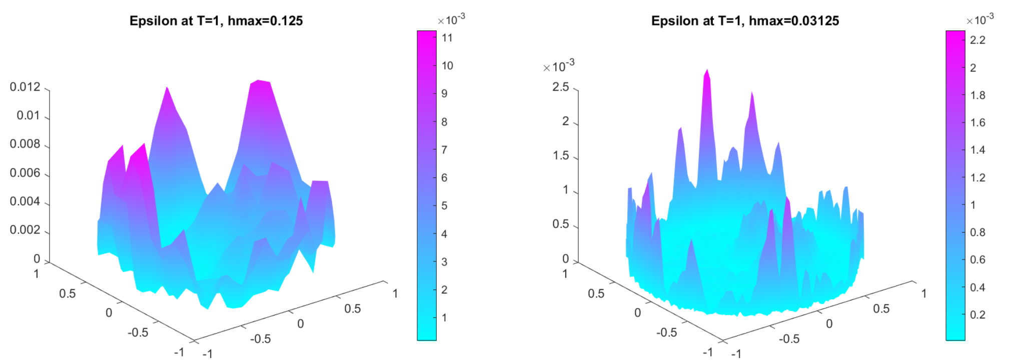
(d) Solution for $hmax = 1/32$

Figure 19: RV solutions of the PDE, non-smooth initial data



(a) Error for $hmax = 1/4$ (b) Error for $hmax = 1/32$

Figure 20: RV, error of the PDE, non-smooth initial data



(a) $\epsilon(U_n), T = 1$ for $hmax = 1/4$ (b) $\epsilon(U_n), T = 1$ for $hmax = 1/32$

Figure 21: Value of $\epsilon(U)$ at the end of the RV simulation, non-smooth initial data

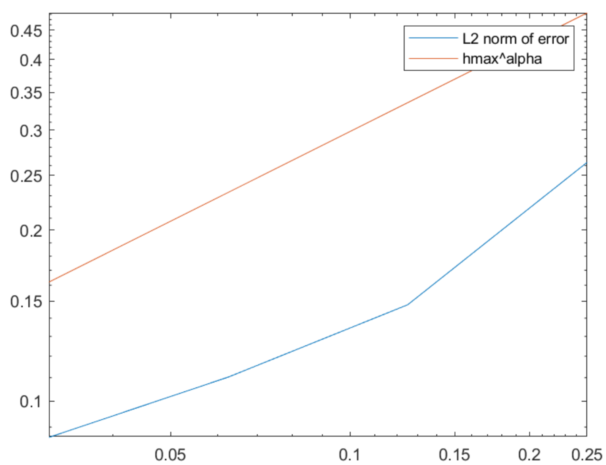


Figure 22: $\alpha = 0.53$ convergence rate with RV method, non-smooth initial data

Summary

With this we concluded the analysis and the numerical modelling of the given tasks. We investigated and implemented the basic FEm, the SUPG and RV method to solve the problem. What we obtained is that the FEM works better with smooth initial data, than with the noncontinuous, resulting better measured convergence rate and much less error. These solutions are numerically not stable, therefore we wanted to use methods, where stability can be proven. We implemented the SUPG and RV methods, which are stable and results in better convergence rate and better solutions at least for the non-smooth data the results are much better with these methods, while for the smooth data the FEM worked just fine.

Appendix

Matlab code for parts $1.1 - 1.3$: *fem.m*

```
%% Given parameters
T=1;
CFL = 0.5;
r0 = 0.25;
x1_0 = 0.3;
x2_0 = 0;

%Used geometry
g = @circleg ;



%% Problem 1 - numerical fem solutions and error + convergence rate calculation
%hmax values to run the simulation for
hmax_list = [1/4, 1/8, 1/16, 1/32];
initial_data = [1, 2];

%Iteration on possible hmax values

for init = initial_data
i=1;
for hmax = hmax_list
    %mesh data
    [p ,e , t ] = initmesh (g , 'hmax' , hmax); % function call to create mesh
    x1=p(1,:)'; % vector of x1 coordinates
    x2=p(2,:)'; % vector of x2 coordinates

    % Set up the matrices
    df=ddf(x1,x2)';
    M = MassAssembler2D(p,t);
    C = ConvectionAssembler2D(p,t,df(1,:),df(2,:));

```

```
%Initial setup

if init==1
U_0 = 1 / 2 * (1 - tanh((((x1 - x1_0).^2 + (x2 - x2_0).^2) / (r0^2)) - 1));
elseif init==2
U_00 = ((x1 - x1_0).^2 + (x2 - x2_0).^2) <r0^2;
U_0=double(U_00);
end

U=U_0;

%Time integration Crank-Nicholson
time = 0;
inf_norm = max(abs(df(1,:))+abs(df(2,:)));
kn = CFL * hmax/inf_norm*1.0; % timstep - not depends on U_n due to f'

while time<T
    %Set up equation matrix / vector
    A=((M/kn) + (C/2));
    b=((M/kn) * U - (C/2) * U);

    %Boundary conditions
    I = eye( length ( p ));
    A(e(1,:) ,:) =I(e(1,:) ,:);
    b(e(1,:)) = 0;

    %Solve equation
    U = A\b;

    time = time + kn;
end


%Plot solution
figure()
pdeplot(p,e,t,"XYData",U,'Zdata',U);
```

```
    title("Solution at T="+T+", hmax="+hmax)

    %Error
    error=U-U_0;
    L2E(i)=sqrt(error'*M*error);
    figure()
    pdeplot (p ,[] ,t,'XYData',error,'ZData',error,'ColorBar','on')
    title("Error at T="+T+", hmax="+hmax)

    i=i+1;
end

% Plot Conv
p = polyfit(log(hmax_list),log(L2E),1);
alpha = p(1)
figure ()
%xlabel ('hmax')
%ylabel('L2E')
loglog( hmax_list , L2E , 'DisplayName' , 'L2 norm of error')
hold on
loglog( hmax_list , hmax_list.^alpha , 'DisplayName' , 'hmax\^alpha')
hold off
legend show

end






%% Assembler functions based on book (The Finite Element Method:Theory, Implementation, and Applications %%%%%%%%%%%

```

```matlab
% derivative f
function df = ddf(x1, x2)
    df = 2 * pi * [-x2, x1];
end

% Convection Matrix Assembler
function C = ConvectionAssembler2D(p,t,bx,by)
    np=size(p,2);
    nt=size(t,2);
    C=sparse(np,np);
    for i=1:nt
        loc2glb=t(1:3,i);
        x=p(1,loc2glb);
        y=p(2,loc2glb);
        [area,b,c]=HatGradients(x,y);
        bxmid=mean(bx(loc2glb));
        bymid=mean(by(loc2glb));
        CK=ones(3,1)*(bxmid*b+bymid*c)'*area/3;
        C(loc2glb,loc2glb)=C(loc2glb,loc2glb)+CK;
    end
end

%Mass Matrix Assembler
function M = MassAssembler2D(p,t)
    np = size(p,2); % number of nodes
    nt = size(t,2); % number of elements
    M = sparse(np,np); % allocate mass matrix
    for K = 1:nt % loop over elements
        loc2glb = t(1:3,K); % local-to-global map
        x = p(1,loc2glb); % node x-coordinates
        y = p(2,loc2glb); % y
        area = polyarea(x,y); % triangle area
        MK = [2, 1, 1; 1, 2, 1; 1, 1, 2]/12*area; %
            element mass matrix
        M(loc2glb,loc2glb) = M(loc2glb,loc2glb)+ MK; %
            add element masses to M
    end
end
```

```
%Gradients
function [area,b,c] = HatGradients(x,y)
area=polyarea(x,y);
b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/area;
c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/area;
end
```

Matlab code for SUPG method: ***SUPG.m***

```
%% Given parameters
T=1;
CFL = 0.5;
r0 = 0.25;
x1_0 = 0.3;
x2_0 = 0;


%Used geometry
g = @circleg ;



%% Problem 1 - numerical fem solutions and error + convergence rate calculation
%hmax values to run the simulation for
hmax_list = [1/4, 1/8, 1/16, 1/32];
initial_data = [1, 2];

%Iteration on possible hmax values

for init = initial_data
i=1;
for hmax = hmax_list
    %mesh data
    delta=0.05*hmax;
    [p ,e , t ] = initmesh (g , 'hmax' , hmax); % function call to create mesh
    x1=p(1,:)'; % vector of x1 coordinates
    x2=p(2,:)'; % vector of x2 coordinates

    % Set up the matrices
    df=ddf(x1,x2)';
    M = MassAssembler2D(p,t);
    C = ConvectionAssembler2D(p,t,df(1,:),df(2,:));
    SC = SCAssembler2D(p,t,df(1,:),df(2,:),delta);
```

```
SM = delta*C';

%Initial setup

 if init==1
 U_0 = 1 / 2 * (1 - tanh((((x1 - x1_0).^2 + (x2 -
    x2_0).^2) / (r0^2)) - 1));
 elseif init==2
 U_00 = ((x1 - x1_0).^2 + (x2 - x2_0).^2) <r0^2;
 U_0=double(U_00);
 end

U=U_0;

%Time integration Crank-Nicholson
 time = 0;
 inf_norm = max(abs(df(1,:))+abs(df(2,:)));
 kn = CFL * hmax/inf_norm*1.0; % timstep - not
    depends on U_n due to f'

 while time<T
     %Set up equatddion matrix / vector
     A=((M/kn) + (C/2) + (SM/kn) + (SC/2));
     b=((M/kn) - (C/2) + (SM/kn) - (SC/2)) * U;

     %Boundary conditions
     I = eye( length ( p ));
     A(e(1,:) ,:) =I(e(1,:) ,:);
     b(e(1,:)) = 0;

     %Solve equation
     U = A\b;

     time = time + kn;
 end


%Plot solution
```

```
    figure()
    pdeplot(p,e,t,"XYData",U,'Zdata',U);
    title("Solution at T="+T+", hmax="+hmax)

   %Error
    error=U-U_0;
   L2E(i)=sqrt(error'*M*error);
    figure()
    pdeplot (p ,[],t,'XYData',error,'ZData',error,'
       ColorBar','on')
    title("Error at T="+T+", hmax="+hmax)

    i=i+1;
end

% Plot Conv
p = polyfit(log(hmax_list),log(L2E),1);
alpha = p(1)
figure ()
%xlabel ('hmax')
%ylabel('L2E')
loglog( hmax_list , L2E , 'DisplayName' , 'L2 norm of
    error')
hold on
loglog( hmax_list , hmax_list.^alpha , 'DisplayName' ,
    'hmax\^alpha')
hold off
legend show

end






%% Assembler functions based on book (The Finite
    Element Method:Theory, Implementation, and
```

```
    Applications %%%%%%%%%%%%%

% derivative f
function df = ddf(x1, x2)
    df = 2 * pi * [-x2, x1];
end

% Convection Matrix Assembler
function C = ConvectionAssembler2D(p,t,bx,by)
    np=size(p,2);
    nt=size(t,2);
    C=sparse(np,np);
    for i=1:nt
        loc2glb=t(1:3,i);
        x=p(1,loc2glb);
        y=p(2,loc2glb);
        [area,b,c]=HatGradients(x,y);
        bxmid=mean(bx(loc2glb));
        bymid=mean(by(loc2glb));
        CK=ones(3,1)*(bxmid*b+bymid*c)'*area/3;
        C(loc2glb,loc2glb)=C(loc2glb,loc2glb)+CK;
    end
end

%Mass Matrix Assembler
function M = MassAssembler2D(p,t)
    np = size(p,2); % number of nodes
    nt = size(t,2); % number of elements
    M = sparse(np,np); % allocate mass matrix
    for K = 1:nt % loop over elements
        loc2glb = t(1:3,K); % local-to-global map
        x = p(1,loc2glb); % node x-coordinates
        y = p(2,loc2glb); % y
        area = polyarea(x,y); % triangle area
        MK = [2, 1, 1; 1, 2, 1; 1, 1, 2]/12*area; %
            element mass matrix
        M(loc2glb,loc2glb) = M(loc2glb,loc2glb)+ MK; %
            add element masses to M
```

```
    end
end

% SC matrix assembly
function A = SCAssembler2D(p,t,bx, by, delta)
np = size(p,2);
nt = size(t,2);
A = sparse(np,np); % allocate stiffness matrix
for K = 1:nt
loc2glb = t(1:3,K); % local-to-global map
x = p(1,loc2glb); % node x-coordinates
y = p(2,loc2glb); % node y-
[area,b,c] = HatGradients(x,y);
bxmid=mean(bx(loc2glb));
bymid=mean(by(loc2glb));
AK = delta*(bxmid^2*b*b'+bymid^2*c*c'+bxmid*bymid*(b*c'+c*b'))*area; % element stiffness matrix
A(loc2glb,loc2glb) = A(loc2glb,loc2glb)+ AK; % add element stiffnesses to A
end
end

%Gradients
function [area,b,c] = HatGradients(x,y)
area=polyarea(x,y);
b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/area;
c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/area;
end
```

Matlab code for RV method: *RV.m*

```
%% Given parameters
T=1;
CFL = 0.5;
r0 = 0.25;
x1_0 = 0.3;
x2_0 = 0;
C_vel=0.25;
C_RV=1;


%Used geometry
g = @circleg ;



%% Problem 1 - numerical fem solutions and error + convergence rate calculation
%hmax values to run the simulation for
hmax_list = [1/4,1/8,1/16,1/32];
initial_data = [1,2];

%Iteration on possible hmax values

for init = initial_data
i=1;L2E=[];
for hmax = hmax_list
     disp("hmax: "+hmax);
     %mesh data
     [p ,e , t ] = initmesh (g , 'hmax' , hmax); % function call to create mesh

     x1=p(1,:)'; % vector of x1 coordinates
     x2=p(2,:)'; % vector of x2 coordinates

     % Set up the matrices
     df=ddf(x1,x2)';
```

```
M = MassAssembler2D(p,t);
C = ConvectionAssembler2D(p,t,df(1,:),df(2,:));

nt = size(t,2); hk=[]; bk=[];
for K = 1:nt % for each element
    loc2glb = t(1:3,K); % local-to-global map
    x = p(1,loc2glb); % node x-coordinates
    y = p(2,loc2glb); % node y-coordinates
    hk(K)=min([norm([x(1)-x(2),y(1)-y(2)]),norm([x(2)-x(3),y(2)-y(3)]),norm([x(3)-x(1),y(3)-y(1)]), 2*norm([x(1)-sum(x)/3,y(1)-sum(y)/3])]); % diameter
    dx=df(1,loc2glb);
    dy=df(2,loc2glb);
    bk(K)=max([norm([dx(1),dy(1)]),norm([dx(2),dy(2)]),norm([dx(3),dy(3)])]);
end

%Initial setup

if init==1
U_0 = 1 / 2 * (1 - tanh((((x1 - x1_0).^2 + (x2 - x2_0).^2) / (r0^2)) - 1));
elseif init==2
U_00 = ((x1 - x1_0).^2 + (x2 - x2_0).^2) <r0^2;
U_0=double(U_00);
end

U=U_0;
U_old=U_0;

%Time integration Crank-Nicholson
time = 0;
inf_norm = max(abs(df(1,:))+abs(df(2,:)));
kn = CFL * hmax/inf_norm*1.0; % timstep - not depends on U_n due to f'

while time<T
```

```
        %disp(time);
        R = ResidualAssembler(p,t, bk, hk, U, U_old, M,
            C, kn,C_vel,C_RV);
        %Set up equatddion matrix / vector
        A=((M/kn) + (C/2) + (R/2));
        b=((M/kn) - (C/2) - (R/2)) * U;

        %Boundary conditions
        I = eye( length ( p ));
        A(e(1,:)  ,:) =I(e(1,:)  ,:);
        b(e(1,:)) = 0;

        %Solve equation
        U_old = U;
        U = A\b;

        time = time + kn;
    end


    %Plot solution
    plt=figure();
    pdeplot(p,e,t,"XYData",U,'Zdata',U);
    title("Solution at T="+T+", hmax="+hmax)
    saveas(plt,"RV_plot_"+init+"_"+hmax+".png")

    %Error
    error=U-U_0;
    L2E(i)=sqrt(error'*M*error);
    err=figure();
    pdeplot (p  ,[],t,'XYData',error,'ZData',error,'
       ColorBar','on')
    title("Error at T="+T+", hmax="+hmax)
    saveas(err,"RV_err_"+init+"_"+hmax+".png")


    %Plot epsilon U_n
    np = size(p,2);
```

```
    nt = size(t,2);
    eps = zeros(np); % allocate matrix
    occurances = zeros(np);
    RU=M\(1/kn*M*(U-U_old)+C*U);
    for K = 1:nt % for each element
    loc2glb = t(1:3,K); % local-to-global map
    x = p(1,loc2glb); % node x-coordinates
    y = p(2,loc2glb); % node y-coordinates
    [area,b,c] = HatGradients(x,y);
    h_k=hk(K);
    b_k=bk(K);
    U_norm=max(abs(U-mean(U)));
    Rk=max(abs(RU(loc2glb)));
    ep=min([C_vel*h_k*b_k,C_RV*h_k^2*Rk/U_norm]);
    eps(loc2glb) = eps(loc2glb)+ 1/3*ep*[1;1;1]; % add element stiffnesses to A
    occurances(loc2glb(1)) = occurances(loc2glb(1))+1;
    occurances(loc2glb(2)) = occurances(loc2glb(2))+1;
    occurances(loc2glb(3)) = occurances(loc2glb(3))+1;
    end
    eps=eps./occurances;
    pdeplot (p ,[],t,'XYData',eps,'ZData',eps,'ColorBar','on')
    title("Epsilon at T="+T+", hmax="+hmax)
    saveas(err,"RV_eps_"+init+"_"+hmax+".png")

    i=i+1;
end

% Plot Conv
p = polyfit(log(hmax_list),log(L2E),1);
alpha = p(1)
alp=figure ();
%xlabel ('hmax')
%ylabel('L2E')
loglog( hmax_list , L2E , 'DisplayName' , 'L2 norm of error')
hold on
```

```
loglog( hmax_list , hmax_list.^alpha , 'DisplayName' ,
    'hmax\^alpha')
hold off
legend show
    saveas(alp, "RV_conv_"+init+".png")

end






%% Assembler functions based on book (The Finite
    Element Method:Theory, Implementation, and
    Applications %%%%%%%%%%%%


% Convection Matrix Assembler
function C = ConvectionAssembler2D(p,t,bx,by)
    np=size(p,2);
    nt=size(t,2);
    C=sparse(np,np);
    for i=1:nt
        loc2glb=t(1:3,i);
        x=p(1,loc2glb);
        y=p(2,loc2glb);
        [area,b,c]=HatGradients(x,y);
        bxmid=mean(bx(loc2glb));
        bymid=mean(by(loc2glb));
        CK=ones(3,1)*(bxmid*b+bymid*c)'*area/3;
        C(loc2glb,loc2glb)=C(loc2glb,loc2glb)+CK;
    end
end

%Mass Matrix Assembler
function M = MassAssembler2D(p,t)
    np = size(p,2); % number of nodes
```

```
    nt = size(t,2); % number of elements
    M = sparse(np,np); % allocate mass matrix
    for K = 1:nt % loop over elements
        loc2glb = t(1:3,K); % local-to-global map
        x = p(1,loc2glb); % node x-coordinates
        y = p(2,loc2glb); % y
        area = polyarea(x,y); % triangle area
        MK = [2, 1, 1; 1, 2, 1; 1, 1, 2]/12*area; %
            element mass matrix
        M(loc2glb,loc2glb) = M(loc2glb,loc2glb)+ MK; %
            add element masses to M
    end
end

% SC matrix assembly
function R = ResidualAssembler(p,t, bk,hk, U, U_old, M,
    C, kn, C_vel,C_RV)
np = size(p,2);
nt = size(t,2);
R = sparse(np,np); % allocate matrix
RU=M\(1/kn*M*(U-U_old)+C*U);
for K = 1:nt % for each element
loc2glb = t(1:3,K); % local-to-global map
x = p(1,loc2glb); % node x-coordinates
y = p(2,loc2glb); % node y-coordinates
[area,b,c] = HatGradients(x,y);

h_k=hk(K);
b_k=bk(K);
U_norm=max(abs(U-mean(U)));
Rk=max(abs(RU(loc2glb)));

e=min([C_vel*h_k*b_k,C_RV*h_k^2*Rk/U_norm]);
AK = e*(b*b'+c*c')*area; % element stiffness matrix
R(loc2glb,loc2glb) = R(loc2glb,loc2glb)+ AK; % add
    element stiffnesses to A
end
end
```

```
%Gradients
function [area,b,c] = HatGradients(x,y)
area=polyarea(x,y);
b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/area;
c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/area;
end

% derivative f
function df = ddf(x1, x2)
    df = 2 * pi * [-x2, x1];
end
```