UPPSALA
UNIVERSITET

Project FEM

Advanced Numerical Methods 10 credits 1TD050 12001 HT2022

Group members:

Ibrohim Hamoud ibrohimmn@gmail.com

(Group)

Uppsala

March 19, 2024

1 Theoretical background

The following linear advection equation

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

is considered in a unit disk $\mathbf{\Omega} = \{\mathbf{x} : x_1^2 + x_2^2 \leq 1\}$ with homogeneous Dirichlet boundary condition on the boundary $\partial\mathbf{\Omega}$ and initial data $u(\mathbf{x},t) = u_0(\mathbf{x})$.

First a subspace $\chi \subseteq H^1$ is constructed for the weak formulation of Eq. (1)

$$\chi_0 := \{v; ||v||^2 + ||\Delta v||^2 < \infty, v(\mathbf{x},t) = 0 \text{ on } \partial\mathbf{\Omega}\}. \tag{2}$$

Hence, a finite dimensional subspace of χ_h can be given by

$$\chi_{h,0} := \{v; v \in C^0(\mathbf{\Omega}), v|_{k_i} \in P^1(k_i), \forall k \in \mathcal{T}_h, v(\mathbf{x},t) = 0 \text{ on } \partial\mathbf{\Omega}\}. \tag{3}$$

Thus, the Galerkin finite element formulation is given to be

$$\begin{aligned} &\text{Find } u_h \in \chi_{h,0} \text{ such that} \\ &\quad (\partial_t u_h, v) + (\nabla.\mathbf{f}(u), v) = 0 \\ &\qquad\qquad\qquad\qquad \forall v \in \chi_{h,0}. \end{aligned} \tag{4}$$

Rewriting the flux term in non-conservative form i.e , $\nabla.\mathbf{f}(u) = \mathbf{f}^{'}(u).\nabla u$, where $\mathbf{f}^{'}(u) := 2\pi(-x_2, x_1)$, gives the following formulation

$$\begin{aligned} &\text{Find } u_h \in \chi_{h,0} \text{ such that} \\ &\quad (\partial_t u_h, v) + (\mathbf{f}^{'}(u_h).\nabla u_h, v) = 0 \\ &\qquad\qquad\qquad\qquad \forall v \in \chi_{h,0}. \end{aligned} \tag{5}$$

Inserting $u_h = \sum_{N_j \in \mathcal{N}_h} \xi_j \phi_j$ with $\{\phi_i\}|_{N_i \in \mathcal{T}_h}$ being the basis of $\chi_{h,0}$, and ξ_i are the solution's values on the nodal points, gives

$$\sum_{\mathcal{N}_i} \Big(\dot{\xi}_j(\phi_j, \phi_i) + \xi_j(\mathcal{B}_i.\nabla\phi_j, \phi_i) \Big) = 0. \tag{6}$$

Thus, the semi-discrete Galerkin finite element formulation is obtained to be

$$\mathbb{M}\dot{\xi} + \mathcal{B}\mathbb{C}\xi, \tag{7}$$

where $\mathbb{M}$ is the mass matrix, and $\mathbb{C}$ is the convection matrix.

Applying the Crank-Nicolson method for time discretization of Eq.(7) gives the fully discrete Galerkin finite element method, that is

$$\frac{\mathbb{M}}{\Delta T}\Big[\xi^{n+1} - \xi^n\Big] = \frac{1}{2}\Big[- \mathcal{B}\mathbb{C}\xi^{\mathbf{n+1}} - \mathcal{B}\mathbb{C}\xi^{\mathbf{n}}\Big], \tag{8}$$

$$\Rightarrow \frac{\mathbb{M}}{\Delta T}\xi^{n+1} + \frac{\mathcal{B}}{2}\mathbb{C}\xi^{n+1} = \frac{\mathbb{M}}{\Delta T}\xi^n - \frac{\mathcal{B}}{2}\mathbb{C}\xi^n, \tag{9}$$

$$\Rightarrow \Big(\frac{\mathbb{M}}{\Delta T} + \frac{\mathcal{B}}{2}\mathbb{C}\Big)\xi^{n+1} = \Big(\frac{\mathbb{M}}{\Delta T} - \frac{\mathcal{B}}{2}\mathbb{C}\Big)\xi^n, \tag{10}$$

$$\Rightarrow \xi^{n+1} = \Big(\frac{\mathbb{M}}{\Delta T} + \frac{\mathcal{B}}{2}\mathbb{C}\Big)^{-1}\Big(\frac{\mathbb{M}}{\Delta T} - \frac{\mathcal{B}}{2}\mathbb{C}\Big)\xi^{n}, \tag{11}$$

Which can be rewritten to be in the form

$$\mathbb{A}\mathbf{u} = \mathbf{b}. \tag{12}$$

2 Problem 1.1

In this part, the solution of Eq. (1) is approximated with Galerkin finite element method in Eq. (11). Here, the initial data is given to be

$$u_0(\mathbf{x}, 0) = \frac{1}{2}\Big(1 - tanh\Big(\frac{(x_1 - x_1^0)^2 - (x_2 - x_2^0)^2}{r_0^2}\Big)\Big), \tag{13}$$

where $r_0 = 0.25$, and $(x_1^0, x_2^0) = (0.3, 0)$. The time stepping is computed from $\Delta T = \mathrm{CLF}\frac{h_{\max}}{||\mathbf{f}'(u)||_{L_\infty}}$, where $\mathrm{CLF} = 0.5$, and $||\mathbf{f}'(u_h)||_{L_\infty} = \max_{N_i \text{ in } K,\, i=0,1,2}\big([(f_1'(u_h))^2 + (f_2'(u_h))^2]^{\frac{1}{2}}(N_i)\big)$.

The view of the solution for mesh sizes of $h_{max} = \frac{1}{8}$ and $h_{max} = \frac{1}{16}$ is shown, respectively, in (Click here) and (Click here). The simulation is given for a final time $T = 1$.

3 Problem 1.2

The L_2-norm of the error is computed to be

$$||e||_{L_2(\mathbf{\Omega})} = \Big(\int_{\Omega} e^2 d\mathbf{x}\Big)^{\frac{1}{2}}, \tag{14}$$

where $e = u_{\text{exact}-u_u}$. The error satisfies the priori error estimate

$$||e||_{L_2(\mathbf{\Omega})} \leq Ch^{\alpha}||u||_{H^2(\mathbf{\Omega})}. \tag{15}$$

Thus, looking at the logarithm of right hand side is $\alpha Ch||u||_{H^2(\omega)}$. The convergence rate α can be found by plotting the error value in log scale, interpolating, and then taking the slope of the resulting function. The results give the convergence rate a value of 1.6893.

Figure 1 presents the plot of the L_2-norm of the error and h_{max}^{α} as a function of mesh sizes.

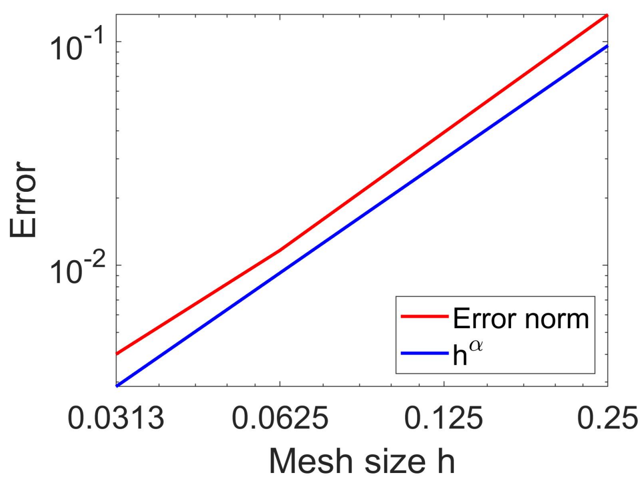


Figure 1: Plots of the L_2-norm of the error and of h^{α} as a function of the mesh size.

4 Problem 1.3

Repeating the above analysis on the following discontinuous initial data

$$u_0(\mathbf{x}) = \begin{cases} 1, & \text{if} \quad (x_1 - x_1^0)^2 + (x_2 - x_2^0)^2 \leq r_0^2, \\ 0, & \text{otherwise}, \end{cases} \tag{16}$$

yields the following simulations (Click here), and (Click here) for $h_{\max=\frac{1}{8}}$ and $h_{\max=\frac{1}{16}}$ respectively. Figure 2 presents the plot of L_2-norm of the error and h^{α}_{max} as a function of the mesh sizes, where *alpha* is found to have the value 0.3228.

Comparing the Galerkin method for discontinuous initial data with the previous continuous initial data in 2, it is observed that the solution starts exhibiting strong oscillations (especially around the discontinuity) even for a more refined case, which contributes to the error observed in figure 2. When it comes to the shape of the solution, it is assumed that the sharp gradient and the discontinuity near the boundary introduce a shape-distortion of the solution that is due to the oscillatory behavior of the Fourier series of the basis function around the discontinuity (Gibbs phenomena).

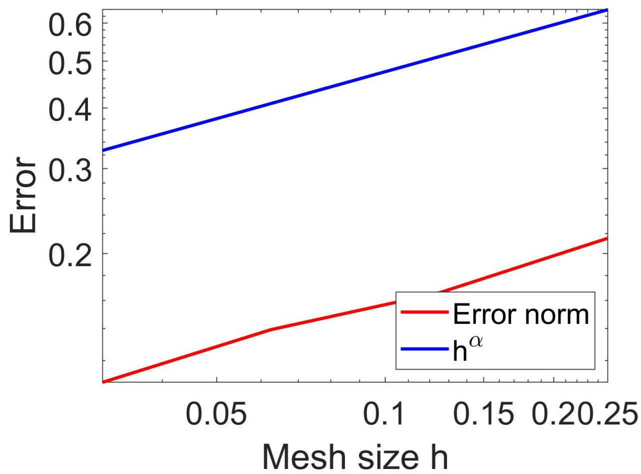


Figure 2: Plots of the L_2-norm of the error and of h^{α} as a function of the mesh size.

5 Part 2

5.1 RV method

To handle the instability occurring in the standard Galerkin finite element method, The residual-based artificial viscosity method is applied to stabilize Eq. (4). Here, a diffusion term that depends on the residual of the solution is added to the standard form in Eq. (4). The new form gives

$$\frac{1}{k^n}(U_n - U_{n-1}) + \frac{1}{2}(\nabla.F_n + \nabla.F_n, v) + \frac{1}{2}(\epsilon_n(\nabla u_n + \nabla u_{n-1}, \nabla v)) = 0, \tag{17}$$

which can be rewritten to be

$$\frac{1}{k^n}(u_h^n, v) + \frac{1}{2}(\mathcal{B}.\nabla u_h^n, v) + \frac{1}{k^n}(\epsilon^n \nabla u_h^n, \nabla v) = \frac{1}{k^n}(u_h^{n-1}, v) - \frac{1}{2}(\mathcal{B}.\nabla u_h^{n-1}, v) - \frac{1}{2}(\epsilon^n \nabla u_h^{n-1}, \nabla v). \tag{18}$$

Thus, the matrix from follows

$$\frac{1}{k^n}\mathbb{M}\xi^n + \frac{1}{2}\mathbb{C}\xi^n + \frac{1}{2}\mathbb{S}\xi^n = \frac{1}{k^n}\mathbb{M}\xi^{n-1} - \frac{1}{2}\mathbb{C}\xi^{n-1} - \frac{1}{2}\mathbb{S}\xi^{n-1}, \tag{19}$$

$$\Rightarrow \big(\frac{1}{k^n}\mathbb{M} + \frac{1}{2}\mathbb{C} + \frac{1}{2}\mathbb{S}\big)\xi^n = \big(\frac{1}{k^n}\mathbb{M} - \frac{1}{2}\mathbb{C} - \frac{1}{2}\mathbb{S}\big)\xi^{n-1}, \tag{20}$$

which has the form

$$\mathbb{A}\mathbf{x} = \mathbf{b}, \tag{21}$$

$$\Rightarrow \xi^n = \big(\frac{1}{k^n}\mathbb{M} + \frac{1}{2}\mathbb{C} + \frac{1}{2}\mathbb{S}\big)^{-1}\big(\frac{1}{k^n}\mathbb{M} - \frac{1}{2}\mathbb{C} - \frac{1}{2}\mathbb{S}\big)\xi^{n-1}. \tag{22}$$

The matrix elements are $\mathbb{M}_{ij} = (\phi_i, \phi_j)$, $\mathbb{C}_{i,j} = (\beta_i.\nabla\phi_i, \phi_j)$ and $\mathbb{S}_{ij} = (\epsilon_K \nabla\phi_i, \nabla\phi_j)$.

The artificial viscosity ϵ is computed for each element from

$$\epsilon_K^n = \min(C_{\text{vel}} h_K \beta_K,\ C_{\text{RV}} h_K^2 \frac{||R||_{\infty,K}}{||u_h^n - \overline{u_h^n}||_{\infty,\Omega}}), \tag{23}$$

where the residual $R(u_h)$ is given by

$$R(u_h^n) = \frac{1}{k^n}(u_h^n - u_h^{n-1}) + \mathcal{B}.\nabla u_h^n, \tag{24}$$

$C_{\text{vel}} = 1$ and $C_{\text{RV}} = 0.25$.

For the continuous boundary condition in Eq. (13) the solution is viewed in (Click here) and (Click here) for $h_{\max} = \frac{1}{8}$ and $h_{\max} = \frac{1}{16}$, respectively. For the discontinuous boundary condition in Eq. (16) the solution is viewed in (Click here) and (Click here) for $h_{\max} = \frac{1}{8}$ and $h_{\max} = \frac{1}{16}$, respectively. When observing the solution for cases with low mesh resolution, it is possible to see the height of the disk diminishing as if it is melting away. This occurrence ceases when the mesh is sufficiently refined. It can be explained that due to the dependency of the viscosity coefficient on h_k in Eq. (23), a large mesh size exacerbates the diffusion affect. This can be observed in the large error for small mesh size in the plot 4 (a) and (c).

5.2 SUPG method

An alternative stabilization technique in given by the Streamline upwind Petrove-Galerkin method (SUPG) which follows

$$(\partial_t u_h + \mathcal{B}.\nabla u_h, v) + \big(\gamma(\partial_t u_h + \mathcal{B}.\nabla u_h), \mathcal{B}.\nabla v\big) = 0. \tag{25}$$

Applying the upwind time-stepping gives

$$\big(\frac{u_h^n - u_h^{n-1}}{k^n}, v\big) + \big(\frac{1}{2}\mathcal{B}.\nabla u_h^n + \frac{1}{2}\mathcal{B}.\nabla u_h^{n-1}, v\big) + \big(\frac{\gamma(u_h^n - u_h^{n-1})}{k^n}, \mathcal{B}.\nabla v\big) + \big(\frac{\gamma\mathcal{B}}{2}.\nabla u_h^n - \frac{\gamma\mathcal{B}}{2}.\nabla u_h^{n-1}, \mathcal{B}.\nabla v\big) = 0, \tag{26}$$

$$\begin{aligned}\Rightarrow \tfrac{1}{k^n}(u_h^n, v) + \tfrac{1}{2}(\mathcal{B}.\nabla u_h^n, v) + \tfrac{1}{k^n}(\gamma u_h^n, \mathcal{B}.v) + \tfrac{1}{2}(\gamma\mathcal{B}.\nabla u_h^n, \mathcal{B}.\nabla v) = \\ \tfrac{1}{k^n}(u_h^{n-1}, v) - \tfrac{1}{2}(\mathcal{B}.\nabla u_h^{n-1}, v) + \tfrac{1}{k^n}(\gamma u_h^{n-1}, \mathcal{B}.v) - \tfrac{1}{2}(\gamma\mathcal{B}.\nabla u_h^{n-1}, \mathcal{B}.\nabla v).\end{aligned} \tag{27}$$

Thus, the matrix form is

$$\Big(\frac{1}{k^n}\mathbb{M}+\frac{1}{2}\mathbb{C}+\frac{\gamma}{k^n}\mathbb{C}^T+\frac{\gamma}{2}\mathbb{S}\Big)\xi^n=\Big(\frac{1}{k^n}\mathbb{M}-\frac{1}{2}\mathbb{C}+\frac{\gamma}{k^n}\mathbb{C}^T-\frac{\gamma}{2}\mathbb{S}\Big)\xi^{n-1}, \tag{28}$$

$$\Rightarrow \xi^n=\Big(\frac{1}{k^n}\mathbb{M}+\frac{1}{2}\mathbb{C}+\frac{\gamma}{k^n}\mathbb{C}^T+\frac{\gamma}{2}\mathbb{S}\Big)^{-1}\Big(\frac{1}{k^n}\mathbb{M}-\frac{1}{2}\mathbb{C}+\frac{\gamma}{k^n}\mathbb{C}^T-\frac{\gamma}{2}\mathbb{S}\Big)\xi^{n-1}. \tag{29}$$

The matrix elements in Eq. (29) are $\mathbb{M}_{ij}=(\phi_i,\phi_j)$, $\mathbb{C}_{i,j}=(\beta_i.\nabla\phi_i,\phi_j)$, $\mathbb{C}^T_{i,j}=(\phi_i,\beta_j.\nabla\phi_j)$ and$\mathbb{S}_{ij}=(\beta_i.\nabla\phi_i,\beta j.\nabla\phi_j)$.

To prove the stability of the SUPG-method, set $v=u_h$ in Eq. (25). thus, follows that (here, u denotes u_h for notation-reduction, and u_n denotes the solution at time-step n)

$$(\partial_t u,u)+(\mathcal{B}.\nabla u,u)+(\partial_t u,\gamma\mathcal{B}.\nabla u)+(\mathcal{B}.\nabla u,\gamma\mathcal{B}.\nabla u)=0. \tag{30}$$

Note that the second term is evaluated into zero. Thus

$$\frac{1}{2}\partial_t||u||^2+(\partial_t u,\gamma,\mathcal{B}.\nabla u)+(\mathcal{B}.\nabla u,\gamma\mathcal{B}.\nabla u)=0 \tag{31}$$

From the PDE $\partial_t u=-\mathcal{B}.\nabla u$. Thus substituting in (31) gives

$$\frac{1}{2}\partial_t||u||^2+(-\mathcal{B}.\nabla u,\gamma\mathcal{B}.\nabla u)+(\mathcal{B}.\nabla u,\gamma\mathcal{B}.\nabla u)=0 \tag{32}$$

$$\frac{1}{2}\partial_t||u||^2=0. \tag{33}$$

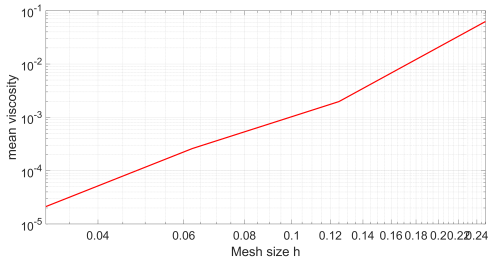
Thus, the method is stable.

For the continuous boundary condition in Eq. (13) the solution is viewed in(Click here) and (Click here) for $h_{\max}=\frac{1}{8}$ and $h_{\max}=\frac{1}{16}$, respectively. For the discontinuous boundary condition in Eq. (16) the solution is viewed in (Click here) and (Click here) for $h_{\max}=\frac{1}{8}$ and $h_{\max}=\frac{1}{16}$, respectively.

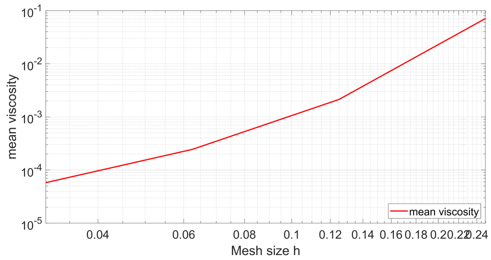
Table 1 shows that the RV-method possesses the best convergence rate for both cases on initial boundary conditions. However, the method is the most expensive computationally and time-consuming. While the GFEM and SUPG are contested in terms of convergence rate. Viewing the simulations for the case of discontinuous boundary condition, The stabilization methods appear to produce more accurate physical behavior compared to the standard-GFEM and the oscillation seems to be gone. The SUPG method still has a section in front of the moving disk which is higher than the expected behavior. From figure 5 (b), it seems that the L_2-error plot does not reflect the advantage of the stabilization method unless the mesh is sufficiently refined. For the continuous boundary condition, no oscillations were observed using GFEM-method. The method is stable in this case and works as intended. Thus, there is no need to employ stabilization techniques and no reason to expect them to score better.

Table 1: Comparison between the convergence rates for the GFEM, RV and SUPG-methods.

Initial condition	GFEM	RV	SUPG
Continuous	1.6893	1.9944	1.5034
Discontinuous	0.3228	0.5400	0.3853

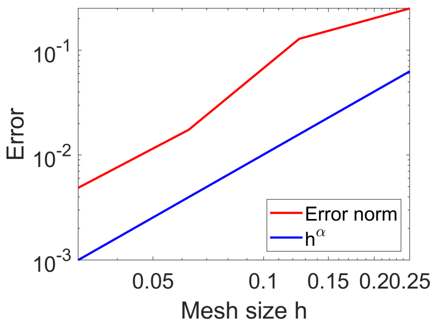


(a) continuous boundary condition.

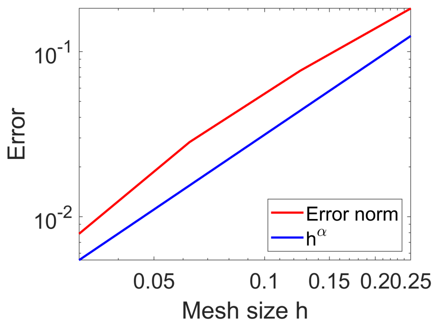


(b) discontinuous boundary condition.

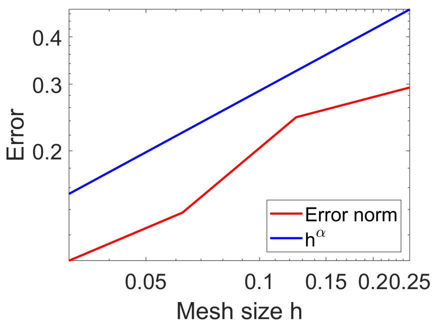
Figure 3: Plots of the mean viscosity in the final solution as function of the mesh-size.



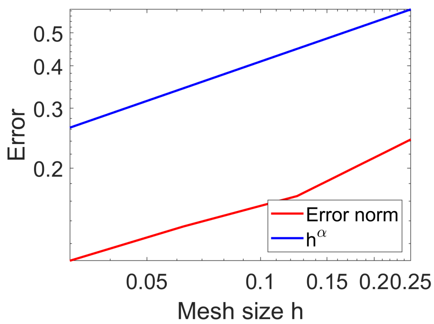
(a) RV-method and continuous boundary condition.



(b) SUPG-method and continuous boundary condition.

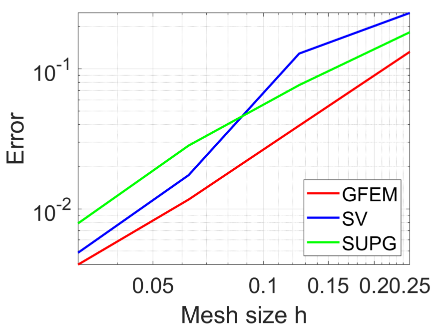


(c) RV-method and discontinuous boundary condition.

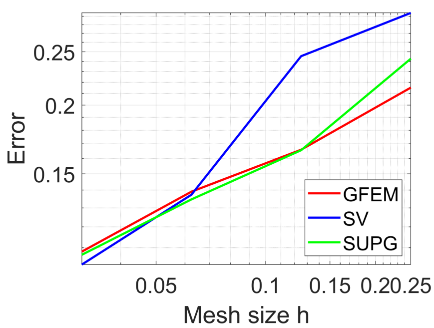


(d) SUPG-method and discontinuous boundary condition.

Figure 4: Plots of the L_2-error and $h_{\max^{\alpha}}$ as function of the mesh-size for the stabilization method RV and SUPG.



(a) continuous boundary condition.



(b) discontinuous boundary condition.

Figure 5: Plots of the L_2-error as function of the mesh-size.

A Source code

A.1 main.m

```
% ((( Uncomment the line to get the results of each task))

% (Problem 1.1)

%ANM_FEM_Part1.GFEM(1/8, "continuous", 1)
%ANM_FEM_Part1GFEM(1/16, "continuous", 1)

%##########################################################################
% (Problem 1.2)

%ANM_FEM_Part1.plot_errors([1/32 1/16 1/8 1/4], "continuous")



%##########################################################################
% (Problem 1.3)

%ANM_FEM_Part1.GFEM(1/8, "discontinuous", 1)
%ANM_FEM_Part1.GFEM(1/16, "discontinuous", 1)
%ANM_FEM_Part1.plot_errors([1/32 1/16 1/8 1/4], "discontinuous")

%###########################################################################
% (Probelm 2.2)

%ANM_FEM_Part2.Solve(1/8, "continuous","RV", 1);
%ANM_FEM_Part2.Solve(1/16, "continuous","RV", 1);


%ANM_FEM_Part2.Solve(1/8, "continuous","SUPG", 1);
%ANM_FEM_Part2.Solve(1/16, "continuous","SUPG", 1);

% to plot the error with h^{\alpha}
ANM_FEM_Part2.plot_errors([1/32 1/16 1/8 1/4], "continuous", "RV")
%ANM_FEM_Part2.plot_errors([1/32 1/16 1/8 1/4], "continuous", "SUPG")


%##########################################################################
% (Probelm 2.3)

%ANM_FEM_Part2.Solve(1/8, "discontinuous","RV", 1);
%ANM_FEM_Part2.Solve(1/16, "discontinuous","RV", 1);

%ANM_FEM_Part2.Solve(1/8, "discontinuous","SUPG", 1);
%ANM_FEM_Part2.Solve(1/16, "discontinuous","SUPG", 1);

% to plot the error with h^{\alpha}
%ANM_FEM_Part2.plot_errors([1/32 1/16 1/8 1/4], "discontinuous", "RV")
%ANM_FEM_Part2.plot_errors([1/32 1/16 1/8 1/4], "discontinuous", "SUPG")
```

A.2 ANM_FEM_Part1.m

```
classdef ANM_FEM_Part1
    methods(Static)
        function CN = GFEM(hmax, command, plotting)
            CFL = 0.5;
            T = 1;

            geometry = @circleg ;
```

```matlab
            [p ,e , t ] = initmesh( geometry, 'hmax' , hmax );
            if command == "continuous"
                u = Functions_part1.initial_u_continuous(p(1,:),p(2,:))';
            else
                u = Functions_part1.initial_u_discrete(p(1,:),p(2,:))';
            end

            ui = u;

            M = Functions_part1.Mass(p,t);
            C = Functions_part1.Convection(p,t);

            I = eye( length ( p ));

            % compute the time-stepping
            F_prime_norm = Functions_part1.L_inf_norm(p);
            k = CFL*hmax/F_prime_norm;

            timing = 0;
            m = round(T/k);
            for i = 1:m

                timing = timing+ k;

                %display(timing)

                A = ( M/k+C/2 );
                b = ( (M/k-C/2) * u );
                A( e(1 ,:) ,:) = I( e(1 ,:) ,:);
                b ( e (1 ,:)) = 0;
                u = A\b;

                if plotting == 1
                    figure(1)
                    pdeplot(p,e,t,'XYData',u,'ZData',u)
                    xlabel('x')
                    ylabel('y')
                    zlabel('u')
                    set(gca,'FontSize',10)
                    %file_name = 'ANM_imagesd16\image' + string(i) + '.png';
                    %saveas(gcf,file_name)
                end

            end
            e = ui - u;
            error = sqrt(e'*M*e);
            CN = error;
            fprintf('For hmax = %0.4f the error =  %f \n\n\n',hmax, error);

        end

        function P = plot_errors(h, command)
            Error = zeros(1,length(h));
            for i = 1:length(h)
                Error(i) = ANM_FEM_Part1.GFEM(h(i), command, 0);
            end

            alpha = polyfit(log(h),log(Error),1);

            %ch = polyval(alpha,h);
            figure(2)
            xticks([1/32, 1/16, 1/8, 1/4])
            loglog(h, Error,'r','LineWidth',2)
            hold on
            loglog(h,h.^alpha(1),'b','LineWidth',2)

            xlabel('Mesh size h','FontSize',15)
```

```
            ylabel('Error','FontSize',15)
            %xlim([0 1/2])
            legend( {'Error norm','h^{\alpha}'} ,'location', 'SouthEast')
            set(gca,'FontSize',20)

        end
    end
end
%###########################################################################
```

A.3 ANM_FEM_Part2.m

```
classdef ANM_FEM_Part2
    methods(Static)
        function CN = Solve(hmax, initial_condition, method, plotting)
            CFL = 0.5;
            T = 1;
            gamma = hmax/2;
            geometry = @circleg ;
            [p ,e , t ] = initmesh( geometry, 'hmax' , hmax );

            if initial_condition == "continuous"
                u = Functions_part2.initial_u_continuous(p(1,:),p(2,:))';
            elseif initial_condition == "discontinuous"
                u = Functions_part2.initial_u_discrete(p(1,:),p(2,:))';
            else
                return
            end

            ui = u;
            M = Functions_part2.Mass(p,t);
            C = Functions_part2.Convection(p,t);
            I = eye( length ( p ));
            % compute the time-stepping
            beta_inf_norm = Functions_part2.Beta_inf_norm(p);
            k = CFL*hmax/beta_inf_norm;
            timing = 0;
            m = round(T/k);
            %m_eps = 0;

            if method == "RV"
                Res = zeros(length(p));
                for i = 1:m

                    timing = timing+ k;
                    %display(timing)
                    epsilons = Functions_part2.Epsilon(p,t,Res, hmax);
                    %m_eps = mean(epsilons);
                    S = Functions_part2.Stiffness(p,t, epsilons);
                    A = ( M/k + C/2 + 1/2*S );
                    b = ( ( M/k - C/2 - 1/2*S) * u );
                    A( e(1 ,:) ,:) = I( e(1 ,:) ,:);
                    b ( e (1 ,:)) = 0;
                    u_prev = u;
                    u = A\b;

                    Res = (1/k*(u - u_prev ) + M\C*u) ;
                    Res = Res/max(u - mean(u));
                    % max(Res)
                    if plotting == 1
                        figure(1)
                        pdeplot(p,e,t,'XYData',u,'ZData',u)
                        xlabel('x')
                        ylabel('y')
```

```
                        zlabel('u')
                        zlim([-0.3,1.2])
                        set(gca,'FontSize',10)
                    %   file_name = 'RV_d_16\image' + string(i) + '.png';
                    %  saveas(gcf,file_name)
                    end


                end


            elseif method == "SUPG"

                S = Functions_part2.SUPG_Stiffness(p,t);
                for i = 1:m

                    timing = timing+ k;
                    %display(timing)
                    A = ( M/k + C/2 + gamma/k*C'  + gamma/2*S );
                    b = ( ( M/k - C/2 + gamma/k*C' - gamma/2*S) * u );
                    A( e(1 ,:) ,:) = I( e(1 ,:) ,:);
                    b ( e (1 ,:)) = 0;
                    u = A\b;

                    if plotting == 1
                        figure(1)
                        pdeplot(p,e,t,'XYData',u,'ZData',u)
                        xlabel('x')
                        ylabel('y')
                        zlabel('u')
                        zlim([-0.3,1.2])
                        set(gca,'FontSize',10)
                        % file_name = 'SUPG_d_16\image' + string(i) + '.png';
                        % saveas(gcf,file_name)
                    end
                end
            else
                return;
            end
            e = ui - u;
            error = sqrt(e'*M*e);
            CN = error;

            fprintf('For hmax = %0.4f the error =  %f \n\n\n',hmax, error);

        end

        function Error = plot_errors(h, initial_conditions, method)
            Error = zeros(1,length(h));
            %eps = zeros(1,length(h));


            for i = 1:length(h)
                disp(i)
                Error(i) = ANM_FEM_Part2.Solve(h(i), initial_conditions,method, 0);
            end


            alpha = polyfit(log(h),log(Error),1);
            disp(alpha)
            %ch = polyval(alpha,h);
            figure(2)
            xticks([1/32, 1/16, 1/8, 1/4])
            loglog(h, Error,'r','LineWidth',2)
            hold on
            loglog(h,h.^alpha(1),'b','LineWidth',2)
            %loglog(h, eps,'r','LineWidth',2)
```

```
            grid on

            xlabel('Mesh size h','FontSize',15)
            ylabel('Error','FontSize',15)
            %xlim([0 1/2])
            %legend( {'mean viscosity'} ,'location', 'SouthEast')

            legend( {'Error norm','h^{\alpha}'} ,'location', 'SouthEast')
            set(gca,'FontSize',20)

        end
    end
end

%################################################################################
```

A.4 Functions_part1.m

```
classdef Functions_part1
    methods(Static)

        %_____________________________________________
        % aAssemble the convection matrix
        function C = Convection(p, t)

           C = sparse(size(p,2),size(p,2));
           for i=1:size(t,2)
               nodes = t(1:3,i);
               x1 = p(1, nodes);
               x2 = p(2, nodes);
               area = polyarea(x1,x2);
               dx2_phi = [x2(2)-x2(3); x2(3)-x2(1); x2(1)-x2(2)]/2/area;
               dx1_phi = [x1(3)-x1(2); x1(1)-x1(3); x1(2)-x1(1)]/2/area;

               C_local = ones(3,1).*(2*pi)*(x1'.*dx1_phi -x2'.*dx2_phi)'* area/3 ;
               C(nodes, nodes) = C(nodes, nodes) + C_local;

           end

        end
        %__________________________________________________________________
        % Assemble the mass matrix
        function M = Mass(p,t)             %t:matrix(4*triangel_number) the first 3 ...
            are corner points and 4 is subdomain number
                                           %p:matrix(2*nodes_number) 1 is x and 2 ...
                                               is y
                                           % size(t,2): triangel number,
                                           % size(p,2): point number
            M = sparse(size(p,2),size(p,2)); % matrix of all the nodes
            for K = 1:size(t,2)            % iterate over all the triangels,
                nodes = t(1:3,K);          % define the node K
                x1 = p(1,nodes);           % node x-coordinate of the node K
                x2 = p(2,nodes);           % node y- coorinate of the node K

                area=polyarea(x1,x2);
                M_local = [2 1 1; 1 2 1; 1 1 2]*area/12;
                M(nodes,nodes) = M(nodes,nodes)+ M_local;
            end
        end
        %__________________________________________________________________
        % Compute ||F'(U)||_(omega )for the CLF
        function norm = L_inf_norm(p)
            i = 1;
            ls = zeros(1,size(p,2));
```

```
        for k = 1:size(p,2)

            x1 = p(1, k);
            x2 = p(2, k);

            f1 = 2*pi* x1;
            f2 = -2*pi * x2;
            ls(i) = (f1^2 + f2^2)^(0.5);
            i = i+1;
        end
        norm = max(ls);
    end
    %___________________________________________________________________
    % Initial continuous data
    function ID = initial_u_continuous(x1,x2)
        r = 0.25;
        x1_0 = 0.3;
        x2_0 = 0;
        ID = 0.5* (1 - tanh(( (x1-x1_0).^2 + (x2-x2_0).^2 )/r^2 - 1));
    end
    %___________________________________________________________________
    % Initial discontinuous data
    function ID = initial_u_discrete(x1,x2)
        r = 0.25;
        x1_0 = 0.3;
        x2_0 = 0;
        ID = zeros(1,length(x1));
        for i = 1:length(x1)
            if ( (x1(i)-x1_0)^2 + (x2(i)-x2_0)^2 ) <= r^2
                ID(i) = 1;
            end
        end

    end
    %___________________________________________________________________
  end
end
```

A.5 Functions_part2.m

```
classdef Functions_part2
  methods(Static)

    %_____________________________________________________
    function A = Stiffness(p,t,eps)

        A = sparse(size(p,2),size(p,2));
        for K = 1:size(t,2)
            nodes = t(1:3,K);
            x2 = p(1,nodes);
            x1 = p(2,nodes);
            area=polyarea(x1,x2);
            epsilon = eps(K);
            dx2_phi = [x2(2)-x2(3); x2(3)-x2(1); x2(1)-x2(2)]/2/area;
            dx1_phi = [x1(3)-x1(2); x1(1)-x1(3); x1(2)-x1(1)]/2/area;


            AK = epsilon*(dx1_phi*dx1_phi'+dx2_phi*dx2_phi')*area;
            A(nodes,nodes) = A(nodes,nodes)+ AK;
        end
    end
    %______________________________________________________________

```

```
function A = SUPG_Stiffness(p,t)

    A = sparse(size(p,2),size(p,2));
    for K = 1:size(t,2)
        nodes = t(1:3,K);
        x2 = p(1,nodes);
        x1 = p(2,nodes);
        beta1 = 2*pi.*x1;
        beta2 = -2*pi.*x2;
        area=polyarea(x1,x2);

        dx2_phi = [x2(2)-x2(3); x2(3)-x2(1); x2(1)-x2(2)]/2/area;
        dx1_phi = [x1(3)-x1(2); x1(1)-x1(3); x1(2)-x1(1)]/2/area;


        AK = (beta1'.*dx1_phi+ ...
            beta2'.*dx2_phi).*(beta1'.*dx1_phi+beta2'.*dx2_phi)'*area;
        A(nodes,nodes) = A(nodes,nodes)+ AK;
    end
end
%___________________________________________________________________
% aAssemble the convection matrix
function C = Convection(p, t)

   C = sparse(size(p,2),size(p,2));
   for i=1:size(t,2)
       nodes = t(1:3,i);
       x1 = p(1, nodes);
       x2 = p(2, nodes);
       area = polyarea(x1,x2);
       dx2_phi = [x2(2)-x2(3); x2(3)-x2(1); x2(1)-x2(2)]/2/area;
       dx1_phi = [x1(3)-x1(2); x1(1)-x1(3); x1(2)-x1(1)]/2/area;

       C_local = ones(3,1).*(2*pi)*(x1'.*dx1_phi -x2'.*dx2_phi)'* area/3 ;
      C(nodes, nodes) = C(nodes, nodes) + C_local;

   end

end
%_________________________________________________________________________
% Assemble the mass matrix
function M = Mass(p,t)                %t:matrix(4*triangel_number) the first 3 ...
    are corner points and 4 is subdomain number
                                      %p:matrix(2*nodes_number) 1 is x and 2 ...
                                          is y
                                      % size(t,2): triangel number,
                                      % size(p,2): point number
    M = sparse(size(p,2),size(p,2)); % matrix of all the nodes
    for K = 1:size(t,2)               % iterate over all the triangels,
        nodes = t(1:3,K);             % define the node K
        x1 = p(1,nodes);              % node x-coordinate of the node K
        x2 = p(2,nodes);              % node y- coorinate of the node K

        area=polyarea(x1,x2);
        M_local = [2 1 1; 1 2 1; 1 1 2]*area/12;
       M(nodes,nodes) = M(nodes,nodes)+ M_local;
    end
end
%_________________________________________________________________________
% Compute ||F'(U)||_(omega )for the CLF
function norm = Beta_inf_norm(p)
    i = 1;
    ls = zeros(1,size(p,2));

    for k = 1:size(p,2)

        x1 = p(1, k);
```

```matlab
                x2 = p(2, k);

                f1 = 2*pi* x1;
                f2 = -2*pi * x2;
                ls(i) = (f1^2 + f2^2)^(0.5);
                i = i+1;
            end
            norm = max(ls);
        end
        %______________________________________________________________
        % Res is the normalized residual
        function epsilon = Epsilon(p,t, Res, h )
            C1 = 0.25;
            C2 = 1.0;
            eps = zeros(1,size(t,2));
            for K = 1:size(t,2) % iterate over the elements
                nodes = t(1:3,K);
                x1 =  2*pi *p(1,nodes);
                x2 = -2*pi *p(2,nodes);
                beta_k = max((x1.^2 + x2.^2).^(0.5));
                Res_k = max(Res(nodes));
                eps(K) = min( C1*h*beta_k , C2*h^2*Res_k );
            end
            epsilon = eps;

        end

        %________________________________________________________________
        % Initial continuous data
        function ID = initial_u_continuous(x1,x2)
            r = 0.25;
            x1_0 = 0.3;
            x2_0 = 0;
            ID = 0.5* (1 - tanh(( (x1-x1_0).^2 + (x2-x2_0).^2 )/r^2 - 1));
        end
        %________________________________________________________________
        % Initial discontinuous data
        function ID = initial_u_discrete(x1,x2)
            r = 0.25;
            x1_0 = 0.3;
            x2_0 = 0;
            ID = zeros(1,length(x1));
            for i = 1:length(x1)
                if ( (x1(i)-x1_0)^2 + (x2(i)-x2_0)^2 ) <= r^2
                    ID(i) = 1;
                end
            end

        end
        %________________________________________________________________
    end
end
```