UPPSALA
UNIVERSITET

Nonlinear Project

ADVANCED NUMERICAL METHODS 10 CREDITS 1TD050 12001 HT2022

Group members:

Ibrohim HAMOUD ibrohimmn@gmail.com

UPPSALA

October 31, 2022

1 FDM part

1.1 Task1, 2 and 3

Set $f = u^2$

$$(u, u_t) = \frac{\alpha}{2}(u_x, f) + (1-\alpha)(f_x, u) - (u_x, \epsilon u_x) - \frac{\alpha}{2}uf\Big|_{x_l}^{x_r} - (1-\alpha)uf\Big|_{x_l}^{x_r} + \epsilon u u_x\Big|_{x_l}^{x_r} \tag{1}$$

$(\mathrm{u}_t, u) = -\frac{\alpha}{2}(f_x, u) - (1-\alpha)(u_x, f) - (\epsilon u_x, u_x) + \epsilon u_x u|_{x_l}^{x_r}$

$$\frac{\partial}{\partial t}||u||^2 = [(1-\alpha) - \frac{\alpha}{2}](f_x, u) + [\frac{\alpha}{2} - (1-\alpha)](u_x, f) + BT \tag{2}$$

We demand that the first two terms are equated to zero. Thus, the obtained value is $\alpha = \frac{2}{3}$, and the boundary term gives

$$BT = (-\frac{3}{2}uf + 2\epsilon u u_x)\Big|_{x_l}^{x_r} = u(-\frac{2}{3}u^2 + 2\epsilon u_x)\Big|_{x_l}^{x_r} \tag{3}$$

An obvious boundary condition can be

$$\begin{aligned} u &= g_r(t), \quad \text{at} \;\; x_r \\ u &= g_l(t), \quad \text{at} \;\; x_l \end{aligned} \tag{4}$$

Strong boundary conditions can are given by

$$\begin{aligned} \beta u(u - |u|) - \epsilon u_x &= g_r(t), \quad \text{at} \;\; x_r \\ \beta u(u + |u|) - \epsilon u_x &= g_l(t), \quad \text{at} \;\; x_l \end{aligned} \tag{5}$$

Inserting the strong boundary conditions in the right boundary term yields

$$-u^3 + 3u(\beta u^2 - \beta u|u| - g_r) = -u^3 + 3\beta u^3 - 3\beta u^2|u| - 3ug_r \tag{6}$$

$$\Rightarrow \begin{array}{ll} \text{when} & u < 0 \Rightarrow (-1 + 3\beta + 3\beta)u^2 - 3ug_r \\ \text{when} & u > 0 \Rightarrow (-1 + 3\beta - 3\beta)u^2 - 3ug_r \end{array} \tag{7}$$

Thus, it is required that $(-1 + 3\beta + 3\beta) = 0$ for $x < 0$

Inserting the strong boundary conditions in the leftt boundary term yields

$$u^3 + -3u(\beta u^2 - \beta u|u| - g_r) = u^3 - 3\beta u^3 - 3\beta u^2|u| + 3ug_r \tag{8}$$

$$\Rightarrow \begin{array}{ll} \text{when} & u < 0 \Rightarrow (-1 - 3\beta + 3\beta)u^2 - 3ug_r \\ \text{when} & u > 0 \Rightarrow (-1 - 3\beta - 3\beta)u^2 - 3ug_r \end{array} \tag{9}$$

Thus, it is required that $(1 - 3\beta - 3\beta) = 0$ for $x < 0$

In conclusion, we obtain the necessary value of $\beta = \frac{1}{6}$. Hence, the energy estimate becomes

$$\Rightarrow \begin{array}{ll} \text{when} & u < 0 \Rightarrow \frac{\partial}{\partial t}||u||^2 = -u^3|^{x_r} - 3ug_r + 3ug_l \\ \text{when} & u > 0 \Rightarrow \frac{\partial}{\partial t}||u||^2 = -u^3|^{x_l} - 3ug_r + 3ug_l \end{array} \tag{10}$$

Dropping the first term yields

$$\frac{\partial}{\partial t}||u||^2 \leq -3ug_r + 3ug_l \tag{11}$$

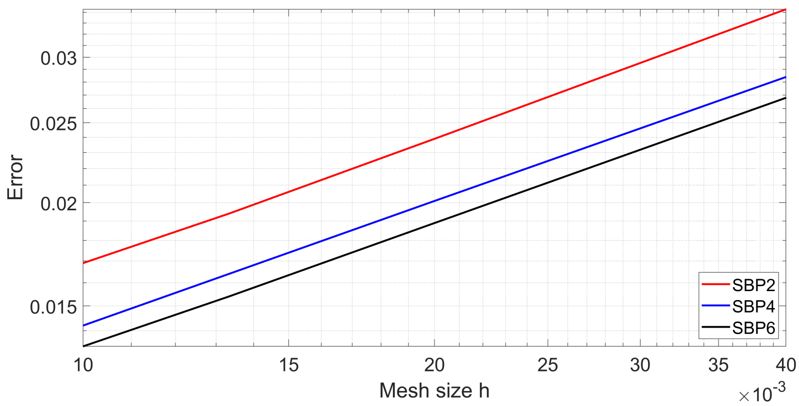
1.2 Task 4

The upwind SBP-projection approximation yields

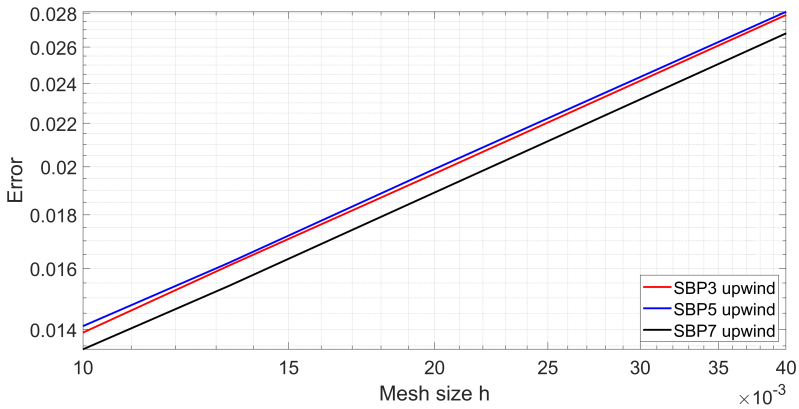
$$\mathbf{V}_t = -\frac{1}{3}\mathbf{D}_-(\bar{\mathbf{W}}\mathbf{W}) - \frac{1}{3}\mathbf{P}\bar{\mathbf{W}}\mathbf{D}_-\mathbf{W} + \mathbf{P}\mathbf{D}_+\bar{v}\mathbf{D}_-\mathbf{W} \tag{12}$$

1.3 Task5

Simulation for the implementation of central difference SBP4 can be viewed here (Click here)central. The implementation for Upwind SBP5 can be viewed here (Click here)upwind. For both simulation, the following parameters hold $t_{\text{final}} = 0.4$, $\epsilon = 0.1$, grid size $m = 101$ and Dirichlet boundary condition imposed. Source code for this task is in appendix A.1.



(a)



(b)

Figure 1: .

1.4 Task6

As ϵ goes to zero, the PDE becomes less stable and spurious oscillation is observed. Therefore, we add artificial dissipation term in the implementation. The simulation for the central difference SBP-projection approximation with the AD-term can be viewed here, for SBP2 (Click here), for SBP4 (Click here), for SBP6 (Click here). In all of the simulation, the following parameters hold $\epsilon = 10^{-6}$, $m = 51$, $t_{\text{final}} = 0.4$ and Dirichlet boundary condition imposed. The γ-parameter that were chosen are displayed in the table 1 along with the L2-error norms. Source code for this task is in appendix A.2.

1.5 Task7

To ensure that the strong stability is not lost with the addition of AD-term, the implementation is repeated with the strong boundary condition imposed instead of Dirichlet's. The simulation for the central difference SBP-projection approximation with the AD-term and strong boundary conditions can be viewed here, for SBP2 (Click here) , for SBP4 (Click here) , for SBP6 (Click here) . In all of the simulation, the following parameters hold $\epsilon = 10^{-6}$, $m = 51$, $t_{\text{final}} = 0.4$. The γ-parameter that were chosen are displayed in the table 1 along with the L2-error norms. Source code for this task is in appendix A.3.

Table 1

Method	SBP2 central	SBP4 central	SBP6 central
Dirichlet BC	0.1220	0.0683	0.0749
Strong BC	0.1220	0.0684	0.0755
γ parameter	0.5	0.4	0.17

1.6 Task8

For this task, the residual viscosity stabilization method is employed. Key term in the implementation is the residual that is computed as follows

$$R(u) = u_t + (\frac{u^2}{2})_x - (\epsilon u_x)_x = D_T(u) + D_f(u) \tag{13}$$

Here, $D_T(u)$ is time dependent term and is approximated by $BDF(u)$ 6th order. In the implementation, the BDF-order is increased from first to sixth order in the first 6 time steps and then it is fixed. Additionally, the viscous term is updated once each time iteration and thus fixed at all RK4 stages. This due to no extra advantage of updating it in each stage and to save computational expenses. The BDF time integrator is not stable for RK4 but is used as no stability requirements are present for BDF.

The viscius term is $\mathbf{D}_2^{(\epsilon)} u$, where $\epsilon = min(\epsilon_1, \epsilon_r)$ and thy are defined as follows:

$$\epsilon_1 = 0.5 * h * max|f^{'}(u)_i| \quad \epsilon_r = h^2 max\frac{|R(u)_i|}{n(u)_i} \tag{14}$$

where $n(u)_i = max(u - \text{mean}(u))$ which is the normalization function and has the same value for all i. In the implementation, no local grid is taken and the Residual and the normalization function is computed over all the domain. The simulation for the implementation can be viewed in (Click here). Here, we use central difference SBP4, $m = 51$, $\epsilon = 10^6$and $t_{\text{final}} = 4$ with the strong boundary condition imposed. The L2-error norm in this implementation is given by 0.0658. Source code for this task is in appendix A.4.

2 FEM part

2.1 Task 2.1

The GFEM that is used in the implementation is given as follows (Here, n denotes time step and k denotes picard's step).

$$(u_{k+1}^{n+1}, v) + \frac{dt}{2}(\beta(u_k^{n+1}).\nabla u_{k+1}^{n-1}, v) = (u^n, v) - \frac{dt}{2}(\beta(u^n).\nabla u^n, v) \tag{15}$$

The simulation can be viewed in (Click here). The source code is in appendix A.5.1. A possible error in the code could be the implementation of the term $\beta(u) = [cos(u), -sin(u)]$. It was a debacle to set it in Fenics in a correct way with no similar implementation in any Fenics tutorial (Line 35 in appendix A.5.1)

The GLS stabilizer that is used is given by the following bilinear and linear forms

$$a(u,v) = (u_{k+1}^{n+1}, v) + \frac{dt}{2}(\beta(u_k^{n+1}).\nabla u_{k+1}^{n+1}, v) + (\gamma u_{k+1}^{n+1}, \beta(u^n.\nabla v)) + (\frac{\gamma dt}{2}\beta(u_k^{n+1}.\nabla u_{k+1}^{n+1}, \beta(u^n).\nabla v)), \tag{16}$$

and the simulation can be viewed in (Click here). The source code is in appendix A.5.2.

The RV stabilization method that is used in the implementation is given by

$$L(v) = (u^n, v) - \frac{dt}{2}(\beta(u^n).\nabla u^n, v) + (\gamma u^n, \beta(u^n).\nabla v) - (\frac{\gamma dt}{2}\beta(u^n).\nabla u^n, \beta(u^n).\nabla v) \tag{17}$$

$$(u_{k+1}^{n+1}, v) + \frac{k}{2}(\beta(u_k^{n+1}).\nabla u_{k+1}^{n+1}, v) + (\hat{\epsilon}\nabla u_{k+1}^{n+1}, \nabla v) = (u^n, v) - \frac{dt}{2}(\beta(u^n).\nabla u^n, v), \tag{18}$$

and the simulation can be viewed in (Click here). In all the implementation the following parameters hold

A Source code

A.1 Source code Task 4 and 5

A.1.1 Central difference method

```
video_on = 0;

% note: Here you dont need PB
m = 51;
x_l = -1 ; x_r = 1;
len = x_r - x_l;
eps= 0.1;
t_end = 0.4;

h=(x_r-x_l)/(m-1);
if video_on
    theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
    scrsz = get(0,'ScreenSize');
    figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
    vidObj = VideoWriter('System');
    open(vidObj);
end

CFL=0.5;
k=CFL*h^(2);
c = ones(m,1);
SBP6_Variable

I = eye(m);
eps_bar = I*eps;


L = [ e_1'; e_m'];
P=I-HI*L'*((L*HI*L')\L);
g = zeros(m,1);
PB =  HI*L'*((L*HI*L')\L);

t=0;
x=linspace(x_l,x_r,m)';
V=zeros(m,1);
V = U_exact(x,0,eps);
freq =0;

max_itter=floor(t_end/k);
for nr_itter=1:max_itter

 %   L = [1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m;
  %       1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1];
  %  P=I-HI*L'*((L*HI*L')\L);

    g(1) = U_exact(x_l,t,eps);
    g(m) = U_exact(x_r,t,eps);
    V0 = P*V+ g;

    w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
        + P*eps_bar*D2*V0 ;
    %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g))   ;%   + Dp*(eps_bar)*Dm*V;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V1 = P*(V+k/2*w1) + g;

    w2=-1/3*P*D1*(diag(V1)*V1)  -1/3*P*diag(V1)*D1*V1 ...
        + P*eps_bar*D2*V1;
```

```
    %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;%   + Dp*(eps_bar)*Dm*V1;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V2 = P*(V+k/2*w2)+ g;

    w3=-1/3*P*D1*(diag(V2)*V2)  -1/3*P*diag(V2)*D1*V2 ...
        + P*eps_bar*D2*V2;
    %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g))  ;% + Dp*(eps_bar)*Dm*V2;

    g(1) = U_exact(x_l,t+k,eps);
    g(m) = U_exact(x_r,t+k,eps);
    V3 = P*(V+k*w3)+ g;
    w4=-1/3*P*D1*(diag(V3)*V3)  -1/3*P*diag(V3)*D1*V3 ...
        + P*eps_bar*D2*V3;
    %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g))  ;% + Dp*(eps_bar)*Dm*V3;
    V=V+k/6*(w1+2*w2+2*w3+w4);

    t=t+k;

    if video_on && (mod(freq,10)==0)
        plot(x,V,'r','LineWidth',1);

       % plot(x,U_exact(x,t),'r','LineWidth',1);
        title(['Numerical solution at t = ',num2str(t)]);
        axis(theAxes);
        grid;xlabel('x');
        legend('v')
        ax = gca;          % current axes
        ax.FontSize = 16;
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
    end
    freq = freq + 1;
end

if video_on
    close(vidObj);
end

U =  U_exact(x,t,eps);
error = sqrt((U-V)'*H*(U-V))


function u_exact = U_exact(x,t,eps)
c = 2;
a = 1;
u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
end
```

A.1.2 Upwind method

```
video_on = 0;


m = 51;
x_l = -1 ; x_r = 1;
len = x_r - x_l;
eps= 0.1;
t_end = 0.4;
freq = 0;
h=(x_r-x_l)/(m-1);
if video_on
    theAxes=[x_l x_r -0.5 3.5];
```

```
    scrsz = get(0,'ScreenSize');
    figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
    vidObj = VideoWriter('System');
    open(vidObj);
end

CFL=0.5;
k=CFL*h^(2);

I = eye(m);
eps_bar = I*eps;
SBP7_Upwind;


L = [ e_1' ;e_m' ];
P=I-HI*L'*((L*HI*L')\L);
g = zeros(m,1);
PB =  HI*L'*((L*HI*L')\L);


t=0;
x=linspace(x_l,x_r,m)';
V=zeros(m,1);
V = U_exact(x,0,eps);

max_itter=floor(t_end/k);
for nr_itter=1:max_itter


     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    g(1) = U_exact(x_l,t,eps);
    g(m) = U_exact(x_r,t,eps);
    V0 = P*V+PB*g;

    w1=-1/3*P*Dm*(diag(V0)*V0) -1/3*P*diag(V0)*Dm*V0 ...
        + P*Dp*eps_bar*Dm*V0;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V1 = P*(V+k/2*w1) +PB*g;

    w2=-1/3*P*Dm*(diag(V1)*V1)  -1/3*P*diag(V1)*Dm*V1...
        + P*Dp*eps_bar*Dm*V1;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V2 = P*(V+k/2*w2)+PB*g;

    w3=-1/3*P*Dm*(diag(V2)*V2)  -1/3*P*diag(V2)*Dm*V2 ...
        + P*Dp*eps_bar*Dm*V2;

    g(1) = U_exact(x_l,t+k,eps);
    g(m) = U_exact(x_r,t+k,eps);
    V3 = P*(V+k*w3)+PB*g;
    w4=-1/3*P*Dm*(diag(V3)*V3)  -1/3*P*diag(V3)*Dm*V3 ...
        + P*Dp*eps_bar*Dm*V3;
    V=V+k/6*(w1+2*w2+2*w3+w4);

    t=t+k;

    if video_on && (mod(freq,10)==0)
        plot(x,V,'r','LineWidth',1);
      % plot(x,U_exact(x,t),'b','LineWidth',1);
        title(['Numerical solution at t = ',num2str(t)]);
        axis(theAxes);
        grid;xlabel('x');
        legend('v')
```

```
            ax = gca;
            ax.FontSize = 16;
            currFrame = getframe;
            writeVideo(vidObj,currFrame);
        end
        freq = freq + 1;
end

if video_on
    close(vidObj);
end

U =  U_exact(x,t,eps);
error = sqrt((U-V)'*H*(U-V))


function u_exact = U_exact(x,t,eps)
c = 2;
a = 1;

u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
end
```

A.2 Source code Task 6

```
video_on = 1;


m = 51;
x_l = -1 ; x_r = 1;
len = x_r - x_l;
eps= 1e-6;
t_end = 0.4;

h=(x_r-x_l)/(m-1);
if video_on
    theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
    scrsz = get(0,'ScreenSize');
    figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
    vidObj = VideoWriter('System');
    open(vidObj);
end

CFL=0.5;
k=CFL*h^(2);

c= ones(m,1);


% Uncomment one of these
%{
SBP6_Variable
gamma = 0.17;
DI = -gamma*HI*DD_3'*DD_3;
%}

%{
SBP4_Variable
gamma = 0.4;
DI = -gamma*HI*DD_2'*DD_2;
%}


SBP2_Variable
```

```
gamma = 0.5;
DI = -gamma*HI*DD_1'*DD_1;

I = eye(m);
eps_bar = I*eps;


L = [ e_1'; e_m'];
P=I-HI*L'*((L*HI*L')\L);
g = zeros(m,1);
PB =  HI*L'*((L*HI*L')\L);

t=0;
x=linspace(x_l,x_r,m)';
V=zeros(m,1);
V = U_exact(x,0,eps);
freq = 0;

max_itter=floor(t_end/k);
for nr_itter=1:max_itter




    g(1) = U_exact(x_l,t,eps);
    g(m) = U_exact(x_r,t,eps);
    V0 = P*V+PB*g;

    w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
        + P*eps_bar*D2*V0 + DI*V0 ;
    %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g))   ;%   + Dp*(eps_bar)*Dm*V;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V1 = P*(V+k/2*w1) +PB*g;

    w2=-1/3*P*D1*(diag(V1)*V1)  -1/3*P*diag(V1)*D1*V1 ...
        + P*eps_bar*D2*V1 +DI*V1;
    %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;%  + Dp*(eps_bar)*Dm*V1;

    g(1) = U_exact(x_l,t+k/2,eps);
    g(m) = U_exact(x_r,t+k/2,eps);
    V2 = P*(V+k/2*w2)+PB*g  ;

    w3=-1/3*P*D1*(diag(V2)*V2)  -1/3*P*diag(V2)*D1*V2 ...
        + P*eps_bar*D2*V2 +DI*V2;
    %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g))  ;% + Dp*(eps_bar)*Dm*V2;

    g(1) = U_exact(x_l,t+k,eps);
    g(m) = U_exact(x_r,t+k,eps);
    V3 = P*(V+k*w3)+PB*g;
    w4=-1/3*P*D1*(diag(V3)*V3)  -1/3*P*diag(V3)*D1*V3 ...
        + P*eps_bar*D2*V3 +DI*V3;
    %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g))  ;% + Dp*(eps_bar)*Dm*V3;
    V=V+k/6*(w1+2*w2+2*w3+w4);

    t=t+k;

    if video_on && (mod(freq,2)==0)

        plot(x,V,'r','LineWidth',1);

       % plot(x,U_exact(x,t),'r','LineWidth',1);
        title(['Numerical solution at t = ',num2str(t)]);
        axis(theAxes);
        grid;xlabel('x');
        legend('v')
```

```
        ax = gca;           % current axes
        ax.FontSize = 16;
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
    end
    freq = freq + 1;
end

if video_on
    close(vidObj);
end

U =  U_exact(x,t,eps);
error = sqrt((U-V)'*H*(U-V))


function u_exact = U_exact(x,t,eps)
c = 2;
a = 1;
u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
end
```

A.3 Source code Task 7

```
video_on = 1;


m = 51;
x_l = -1 ; x_r = 1;
len = x_r - x_l;
eps= 1e-6;
t_end = 0.4;

h=(x_r-x_l)/(m-1);
if video_on
    theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
    scrsz = get(0,'ScreenSize');
    figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
    vidObj = VideoWriter('System');
    open(vidObj);
end

CFL=0.5;
k=CFL*h^(2);

c=ones(m,1);


%Uncomment one of these:
%{
SBP6_Variable
gamma = 0.17;
DI = -gamma*HI*DD_3'*DD_3;
%}

%{
SBP4_Variable
gamma = 0.4;
DI = -gamma*HI*DD_2'*DD_2;
%}


SBP2_Variable
gamma = 0.5;
```

```
DI = -gamma*HI*DD_1'*DD_1;

I = eye(m);
eps_bar = I*eps;


%L = [ e_1'; e_m'];
%P=I-HI*L'*((L*HI*L')\L);
g = zeros(2,1);
%PB =  HI*L'*((L*HI*L')\L);

t=0;
x=linspace(x_l,x_r,m)';
V=zeros(m,1);
V = U_exact(x,0,eps);
freq = 0;

max_itter=floor(t_end/k);
for nr_itter=1:max_itter

    L = [1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1;
         1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m];
    P=I-HI*L'*((L*HI*L')\L);
    %PB =  HI*L'*((L*HI*L')\L);
    %UU = U_exact(x,t,eps);
    var1 = U_exact(x_l,t,eps); var2 = U_exact(x_r,t,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V0 = (P)*V +PB;

    w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
        + P*eps_bar*D2*V0 + DI*V0 ;
    %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g))   ;%   + Dp*(eps_bar)*Dm*V;

    UU = U_exact(x,t+k/2,eps);
    var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t+k/2,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t+k/2,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V1 = (P)*(V+k/2*w1)+PB ;

    w2=-1/3*P*D1*(diag(V1)*V1)  -1/3*P*diag(V1)*D1*V1 ...
        + P*eps_bar*D2*V1 +DI*V1;
    %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;%  + Dp*(eps_bar)*Dm*V1;

    var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k/2,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k/2,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V2 = (P)*(V+k/2*w2)+PB ;

    w3=-1/3*P*D1*(diag(V2)*V2)  -1/3*P*diag(V2)*D1*V2 ...
        + P*eps_bar*D2*V2 +DI*V2;
    %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g))  ;% + Dp*(eps_bar)*Dm*V2;
    UU = U_exact(x,t+k,eps);
    var1 = U_exact(x_l,t+k,eps); var2 = U_exact(x_r,t+k,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V3 = (P)*(V+k*w3) +PB;
    w4=-1/3*P*D1*(diag(V3)*V3)  -1/3*P*diag(V3)*D1*V3 ...
        + P*eps_bar*D2*V3 +DI*V3;
    %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g))  ;% + Dp*(eps_bar)*Dm*V3;
    V=V+k/6*(w1+2*w2+2*w3+w4);

    t=t+k;
```

```

    if video_on && (mod(freq,2)==0)

        plot(x,V,'r','LineWidth',1);

       % plot(x,U_exact(x,t,eps),'r','LineWidth',1);

       % plot(x,U_diff_exact(x,t,eps),'b','LineWidth',1);
        title(['Numerical solution at t = ',num2str(t)]);
        axis(theAxes);
        grid;xlabel('x');
        legend('v')
        ax = gca;            % current axes
        ax.FontSize = 16;
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
    end
    freq = freq + 1;
end

if video_on
    close(vidObj);
end

U =  U_exact(x,t,eps);
error = sqrt((U-V)'*H*(U-V))


function u_exact = U_exact(x,t,eps)
c = 2;
a = 1;
u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
end

function u_diff_exact = U_diff_exact(x,t,eps)
u_diff_exact = -(tanh((2*t - x)/(2*eps)).^2 - 1)/(2*eps);
end
```

A.4 Source code Task 8

```
% BDF1
% This in the file BDF.m
function bdf = BDF(VV,command)
if command == 0
    bdf = 0*VV(7,:);
end
if command == 1
    bdf = VV(7,:) - VV(6,:);
end

if command == 2
    bdf = VV(7,:)-(4/3)*VV(6,:)+(1/3)*VV(5,:);
end

if command == 3
    bdf = VV(7,:)-(18/11)*VV(6,:)+(9/11)*VV(5,:)- ...
        (2/11)*VV(4,:);
end

if command == 4
    bdf = VV(7,:)-(48/25)*VV(6,:)+(36/25)*VV(5,:)- ...
        (16/25)*VV(4,:)+(3/25)*VV(3,:);
end

```

```matlab
if command == 5
    bdf = VV(7,:)-(300/137)*VV(6,:)+(300.137)*VV(5,:)- ...
        (200/137)*VV(4,:)+(75/137)*VV(3,:)-(12/137)*VV(2,:);
end

if command >= 6
    bdf = VV(7,:)-(360/147)*VV(6,:)+(450/147)*VV(5,:) -...
        (400/147)*VV(4,:)+(225/147)*VV(3,:)-(72/147)*VV(2,:)+...
        (10/147)*VV(1,:);
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% in the file Task8.m
video_on = 1;


m = 51;
x_l = -1 ; x_r = 1;
len = x_r - x_l;
eps= 1e-6;
t_end = 0.4;

h=(x_r-x_l)/(m-1);
if video_on
    theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
    scrsz = get(0,'ScreenSize');
    figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
    vidObj = VideoWriter('System');
    open(vidObj);
end

CFL=0.5;
k=CFL*h^(2);



c=ones(m,1);
SBP4_Variable;

I = eye(m);
eps_bar = I*eps;


%L = [ e_1'; e_m'];
%P=I-HI*L'*((L*HI*L')\L);
g = zeros(2,1);
%PB =  HI*L'*((L*HI*L')\L);

t=0;
x=linspace(x_l,x_r,m)';
V=zeros(m,1);
V = U_exact(x,0,eps);
freq = 0;

% store the last 6 solution
VV_store = [0*V';0*V';0*V';0*V';0*V';0*V';V'];

% BDF_command tells which order PDF to use
BDF_command = 0;
Residual = zeros(m,1);
BDF_u = zeros(1,m);



max_itter=floor(t_end/k);
```

```
for nr_itter=1:max_itter

    % BDF time integral for the viscocity
    BDF_u=BDF(VV_store,BDF_command);


    L = [1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1;
        1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m];
    P=I-HI*L'*((L*HI*L')\L);

    % related to RK4 stage1
    var1 = U_exact(x_l,t,eps); var2 = U_exact(x_r,t,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V0 = (P)*V +PB;

    % Compute viscocity
   % c=ones(m,1);
    %D2=HI*(-M-diag(c)*e_1*d_1+diag(c)*e_m*d_m);
    Residual = BDF_u' +1/3*P*D1*(diag(V0)*V0)  +1/3*P*diag(V0)*D1*V0 ...
        - P*D2*V0;

    % we dont use local for start

    eps_r = h^(2)* abs(Residual) ./ ( abs(V0-max(V0-mean(V0)) )      );
    eps_l = 0.5*h *abs(0.5*V0);
    %eps_viscocity
    c = min(eps_l,eps_r);
    SBP4_Variable;


    %%%% RK4 starts here
    % Stage 1
    w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
        + P*D2*V0   ;


    % Stage 2
    var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t+k/2,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t+k/2,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V1 = (P)*(V+k/2*w1)+PB ;

    w2=-1/3*P*D1*(diag(V1)*V1)  -1/3*P*diag(V1)*D1*V1 ...
        + P*D2*V1 ;

    % Stage 3
    var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k/2,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k/2,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V2 = (P)*(V+k/2*w2)+PB ;

    w3=-1/3*P*D1*(diag(V2)*V2)  -1/3*P*diag(V2)*D1*V2 ...
        + P*D2*V2 ;

    % Stage4
    var1 = U_exact(x_l,t+k,eps); var2 = U_exact(x_r,t+k,eps);
    g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k,eps);
    g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k,eps);
    PB =  HI*L'*((L*HI*L')\g);
    V3 = (P)*(V+k*w3) +PB;
    w4=-1/3*P*D1*(diag(V3)*V3)  -1/3*P*diag(V3)*D1*V3 ...
        + P*D2*V3 ;

```

```
    V=V+k/6*(w1+2*w2+2*w3+w4);

    t=t+k;

    if video_on && (mod(freq,2)==0)

        plot(x,V,'r','LineWidth',1);

       % plot(x,U_exact(x,t,eps),'r','LineWidth',1);

       % plot(x,U_diff_exact(x,t,eps),'b','LineWidth',1);
        title(['Numerical solution at t = ',num2str(t)]);
        axis(theAxes);
        grid;xlabel('x');
        legend('v')
        ax = gca;           % current axes
        ax.FontSize = 16;
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
    end
    freq = freq + 1;

    %shift the stored solution to the left and add the new one
    VV_store = circshift(VV_store,-1);
    VV_store(7,:) = V';
    BDF_command = BDF_command +1;

end

if video_on
    close(vidObj);
end

U =  U_exact(x,t,eps);
error = sqrt((U-V)'*H*(U-V))


function u_exact = U_exact(x,t,eps)
c = 2;
a = 1;
u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
end

function u_diff_exact = U_diff_exact(x,t,eps)
u_diff_exact = -(tanh((2*t - x)/(2*eps)).^2 - 1)/(2*eps);
end
```

A.5 Source code Part 2.2

A.5.1 GFEM

```
from dolfin import*
import numpy as np
import scipy.linalg as la
import ufl
import sympy as symp



T = 2
num_samples = 40
dt = 0.05


```

```
mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
V = FunctionSpace(mesh, "CG", 1)
VV = VectorFunctionSpace(mesh,"CG",1)

def boundary(x, on_boundary):
    return on_boundary


bc = DirichletBC(V,Constant(0.0),boundary)

ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2,pi=np.pi)


u0 = interpolate(ID, V)

u = TrialFunction(V)
v = TestFunction(V)
#B = Expression(('sin(u)','cos(u)'),degree=2, u=u)
#B0 = Expression(('sin(u)','cos(u)'), degree=2,u=u0)

def B(u):
   # return Expression(('cos(u)','-sin(u)'),degree=2,u=u)
    return  as_vector(( cos(u), -sin(u) ))


u_k = interpolate(Constant(0.0),V)
eps=1.0
tol = 1.0E-5
maxiter=25



a = u*v*dx + 0.5*dt*dot(B(u_k),grad(u) ) *v*dx
L = +u0*v*dx - 0.5*dt*dot(B(u0),grad(u0) )  *v*dx

#a = u*v*dx + 0.5*dt*div(B(u))
u = Function(V)

out_file = File("VTK/Results.pvd", "compressed")



#u.assign(u0)
t=0
t_save =0.0
out_file << (u0,t)

while t<=T:
    t += dt
    t_save += dt

    itera = 0
    eps =np.Inf

    while eps>tol and itera <maxiter:
        itera += 1

        solve(a==L,u,bc)
        diff = np.array(u.vector()) - np.array(u_k.vector())

        eps = np.linalg.norm(diff, ord=np.Inf)
        u_k.assign(u)

    u0.assign(u)

    if t_save >T/num_samples or t>=T-dt:
        print('time = ',t)
```

```
        out_file << (u,t)
        t_save =0
```

A.5.2 GLS

```
from dolfin import*
import numpy as np
import scipy.linalg as la
import ufl



T = 2
num_samples = 40
dt = 0.05


mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
V = FunctionSpace(mesh, "CG", 1)
VV = VectorFunctionSpace(mesh,"CG",1)

def boundary(x, on_boundary):
    return on_boundary


bc = DirichletBC(V,Constant(0.0),boundary)

ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2,pi=np.pi)


u0 = interpolate(ID, V)

u = TrialFunction(V)
v = TestFunction(V)
#B = Expression(('sin(u)','cos(u)'),degree=2, u=u)
#B0 = Expression(('sin(u)','cos(u)'), degree=2,u=u0)

def B(u):
   # return project(Expression(('-cos(u)','sin(u)'),degree=2,u=u),VV )
    return    as_vector(( cos(u), -sin(u) ))
  # return np.array([sym(cos(u)),sym(sin(u))])


#def f(u):
 #    return as_vector((sin(u),cos(u)))

u_k = interpolate(Constant(0.0),V)
eps=1.0
tol = 1.0E-5
maxiter=25

h = 4/31
gamma = 0.5*h

a = u*v*dx + 0.5*dt*dot(B(u_k),grad(u) ) *v*dx\
        +gamma*u*dot(B(u0),grad(v))*dx \
        +0.5*gamma*dt*dot(B(u_k),grad(u))*dot(B(u0),grad(v))*dx

L = u0*v*dx - 0.5*dt*dot(B(u0),grad(u0) )*v*dx \
        +gamma*u0*dot(B(u0),grad(v))*dx\
        -0.5*gamma*dt*dot(B(u0),grad(u0))*dot(B(u0),grad(v))*dx

u = Function(V)

```

```


out_file = File("VTK/Results.pvd", "compressed")


#u.assign(u0)
t=0
t_save =0.0
out_file << (u0,t)

while t<=T:
    t += dt
    t_save += dt

    itera = 0
    eps =np.Inf

    while eps>tol and itera <maxiter:
        itera += 1

        solve(a==L,u,bc)
        diff = np.array(u.vector()) - np.array(u_k.vector())

        eps = np.linalg.norm(diff, ord=np.Inf)
        u_k.assign(u)

    u0.assign(u)

    if t_save >T/num_samples or t>=T-dt:
        print('time = ',t)
        out_file << (u,t)
        t_save =0
```

A.5.3 RV

```
from dolfin import*
import numpy as np
import scipy.linalg as la
import ufl



T = 2.0
num_samples = 40
dt = 0.05


mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
V = FunctionSpace(mesh, "CG", 1)
VV = VectorFunctionSpace(mesh,"CG",1)

def boundary(x, on_boundary):
    return on_boundary


bc = DirichletBC(V,Constant(0.0),boundary)

ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2,pi=np.pi)


u0 = interpolate(ID, V)

u = TrialFunction(V)
v = TestFunction(V)
```

```
#B = Expression(('sin(u)','cos(u)'),degree=2, u=u)
#B0 = Expression(('sin(u)','cos(u)'), degree=2,u=u0)

def B(u):
   # return project(Expression(('cos(u)','-sin(u)'),degree=2,u=u),VV )
    return    as_vector(( cos(u), -sin(u) ))
  # return np.array([sym(cos(u)),sym(-sin(u))])


#def f(u):
 #   return as_vector((sin(u),cos(u)))

u_k = interpolate(Constant(0.0),V)
eps=1.0
tol = 1.0E-5
maxiter=25

#h = CellDiameter(mesh)
h = 4/30
EPS = interpolate(Constant(0.0),V)

"""
def max_norm(u):
    u_array = np.array(u.vector())
    Max = np.max(u_array - np.mean(u_array))
   # MAX = np.max(u_array - assemble(u*dx)/assemble(1*dx))
    return Max
"""


a = u*v*dx + 0.5*dt*dot(B(u_k),grad(u) ) *v*dx\
        +0.5*dt*dot(EPS*grad(u),grad(v))*dx


L = u0*v*dx - 0.5*dt*dot(B(u0),grad(u0) )*v*dx \
        - 0.5*dt*dot(EPS*grad(u0),grad(v))*dx

u = Function(V)




out_file = File("VTK/Results.pvd", "compressed")


#u.assign(u0)
t=0
t_save =0.0
out_file << (u0,t)

while t<=T:
    t += dt
    t_save += dt

    itera = 0
    eps =np.Inf

    while eps>tol and itera <maxiter:
        itera += 1

        solve(a==L,u,bc)
        diff = np.array(u.vector()) - np.array(u_k.vector())

        eps = np.linalg.norm(diff, ord=np.Inf)
        u_k.assign(u)

    u_array = np.array(u.vector())
```

```
    u0_array = np.array(u0.vector())
    flux = project(dot(B(u),grad(u)),V)
    flux_array = np.array(flux.vector())

    Res = dt*(u_array-u0_array) + flux_array
    beta = norm(project(B(u),VV).vector(),'linf')

    dX = Measure('dx', mesh)
    max_normm = np.max(u_array-assemble(u*dX)/assemble(1*dX))
    epsilons = np.minimum(h*beta,0.25*h**2 *abs(Res)/max_normm)

    EPS.vector()[:] = epsilons
   # EPS.assign(epsilons)


    u0.assign(u)

    if t_save >T/num_samples or t>=T-dt:
        print('time = ',t)
        out_file << (u,t)
        t_save =0
```