



# 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  $f = u^2$

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

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

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

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

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

An obvious boundary condition can be

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

Strong boundary conditions can are given by

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

Inserting the strong boundary conditions in the right boundary term yields

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

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

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

Inserting the strong boundary conditions in the left boundary term yields

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

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

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

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

Dropping the first term yields

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

## 1.2 Task 4

The upwind SBP-projection approximation yields

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

## 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.

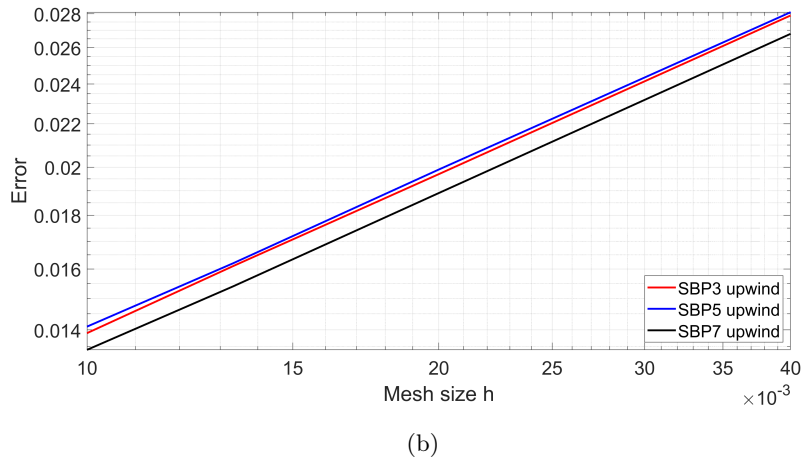
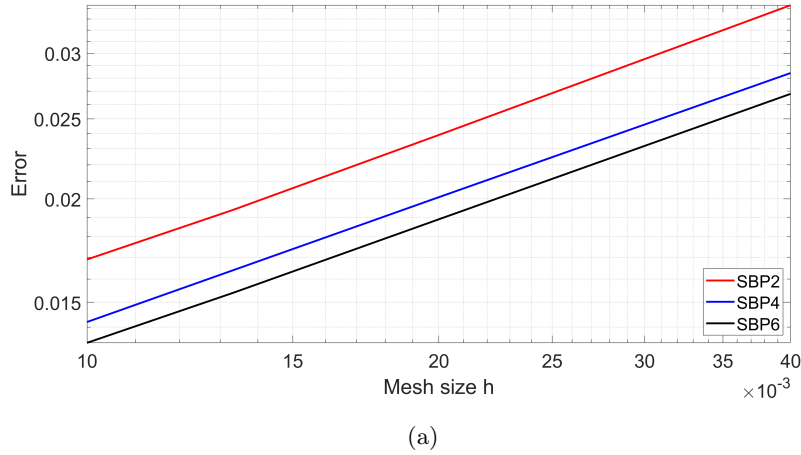


Figure 1: .

## 1.4 Task6

As  $\epsilon$  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  $\gamma$ -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  $\gamma$ -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
$\gamma$ 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

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

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

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

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).

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

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

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

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

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

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

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

```
1  video_on = 0;
2
3  % note: Here you dont need PB
4  m = 51;
5  x_l = -1 ; x_r = 1;
6  len = x_r - x_l;
7  eps = 0.1;
8  t_end = 0.4;
9
10 h=(x_r-x_l)/(m-1);
11 if video_on
12     theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
13     scrsz = get(0, 'ScreenSize');
14     figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
15     vidObj = VideoWriter('System');
16     open(vidObj);
17 end
18
19 CFL=0.5;
20 k=CFL*h^(2);
21 c = ones(m,1);
22 SBP6_Variable
23
24 I = eye(m);
25 eps_bar = I*eps;
26
27
28 L = [ e_1'; e_m'];
29 P=I-HI*L'*((L*HI*L')\L);
30 g = zeros(m,1);
31 PB = HI*L'*((L*HI*L')\L);
32
33 t=0;
34 x=linspace(x_l,x_r,m)';
35 V=zeros(m,1);
36 V = U_exact(x,0,eps);
37 freq =0;
38
39 max_itter=floor(t_end/k);
40 for nr_itter=1:max_itter
41
42     % L = [1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m;
43     %      1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1];
44     % P=I-HI*L'*((L*HI*L')\L);
45
46     g(1) = U_exact(x_l,t,eps);
47     g(m) = U_exact(x_r,t,eps);
48     V0 = P*V+ g;
49
50     w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
51         + P*eps_bar*D2*V0 ;
52     %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g)) ;% + Dp*(eps_bar)*Dm*V;
53
54     g(1) = U_exact(x_l,t+k/2,eps);
55     g(m) = U_exact(x_r,t+k/2,eps);
56     V1 = P*(V+k/2*w1) + g;
57
58     w2=-1/3*P*D1*(diag(V1)*V1) -1/3*P*diag(V1)*D1*V1 ...
59         + P*eps_bar*D2*V1;
```

```

60     %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;% + Dp*(eps_bar)*Dm*V1;
61
62     g(1) = U_exact(x_l,t+k/2,eps);
63     g(m) = U_exact(x_r,t+k/2,eps);
64     V2 = P*(V+k/2*w2)+ g;
65
66     w3=-1/3*P*D1*(diag(V2)*V2) -1/3*P*diag(V2)*D1*V2 ...
67         + P*eps_bar*D2*V2;
68     %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g)) ;% + Dp*(eps_bar)*Dm*V2;
69
70     g(1) = U_exact(x_l,t+k,eps);
71     g(m) = U_exact(x_r,t+k,eps);
72     V3 = P*(V+k*w3)+ g;
73     w4=-1/3*P*D1*(diag(V3)*V3) -1/3*P*diag(V3)*D1*V3 ...
74         + P*eps_bar*D2*V3;
75     %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g)) ;% + Dp*(eps_bar)*Dm*V3;
76     V=V+k/6*(w1+2*w2+2*w3+w4);
77
78     t=t+k;
79
80     if video_on && (mod(freq,10)==0)
81         plot(x,V,'r','LineWidth',1);
82
83         % plot(x,U_exact(x,t),'r','LineWidth',1);
84         title(['Numerical solution at t = ',num2str(t)]);
85         axis(theAxes);
86         grid;xlabel('x');
87         legend('v')
88         ax = gca; % current axes
89         ax.FontSize = 16;
90         currFrame = getframe;
91         writeVideo(vidObj,currFrame);
92     end
93     freq = freq + 1;
94 end
95
96 if video_on
97     close(vidObj);
98 end
99
100 U = U_exact(x,t,eps);
101 error = sqrt((U-V)'*H*(U-V))
102
103
104 function u_exact = U_exact(x,t,eps)
105 c = 2;
106 a = 1;
107 u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
108 end

```

### A.1.2 Upwind method

```

1  video_on = 0;
2
3
4  m = 51;
5  x_l = -1 ; x_r = 1;
6  len = x_r - x_l;
7  eps= 0.1;
8  t_end = 0.4;
9  freq = 0;
10 h=(x_r-x_l)/(m-1);
11 if video_on
12     theAxes=[x_l x_r -0.5 3.5];

```

```

13     scrsz = get(0, 'ScreenSize');
14     figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
15     vidObj = VideoWriter('System');
16     open(vidObj);
17 end
18
19 CFL=0.5;
20 k=CFL*h^(2);
21
22 I = eye(m);
23 eps_bar = I*eps;
24 SBP7_Upwind;
25
26
27 L = [ e_1' ;e_m' ];
28 P=I-HI*L'*((L*HI*L')\L);
29 g = zeros(m,1);
30 PB = HI*L'*((L*HI*L')\L);
31
32
33 t=0;
34 x=linspace(x_l,x_r,m)';
35 V=zeros(m,1);
36 V = U_exact(x,0,eps);
37
38 max_itter=floor(t_end/k);
39 for nr_itter=1:max_itter
40
41
42     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
43     g(1) = U_exact(x_l,t,eps);
44     g(m) = U_exact(x_r,t,eps);
45     V0 = P*V+PB*g;
46
47     w1=-1/3*P*Dm*(diag(V0)*V0) -1/3*P*diag(V0)*Dm*V0 ...
48         + P*Dp*eps_bar*Dm*V0;
49
50     g(1) = U_exact(x_l,t+k/2,eps);
51     g(m) = U_exact(x_r,t+k/2,eps);
52     V1 = P*(V+k/2*w1)+PB*g;
53
54     w2=-1/3*P*Dm*(diag(V1)*V1) -1/3*P*diag(V1)*Dm*V1 ...
55         + P*Dp*eps_bar*Dm*V1;
56
57     g(1) = U_exact(x_l,t+k/2,eps);
58     g(m) = U_exact(x_r,t+k/2,eps);
59     V2 = P*(V+k/2*w2)+PB*g;
60
61     w3=-1/3*P*Dm*(diag(V2)*V2) -1/3*P*diag(V2)*Dm*V2 ...
62         + P*Dp*eps_bar*Dm*V2;
63
64     g(1) = U_exact(x_l,t+k,eps);
65     g(m) = U_exact(x_r,t+k,eps);
66     V3 = P*(V+k*w3)+PB*g;
67     w4=-1/3*P*Dm*(diag(V3)*V3) -1/3*P*diag(V3)*Dm*V3 ...
68         + P*Dp*eps_bar*Dm*V3;
69     V=V+k/6*(w1+2*w2+2*w3+w4);
70
71     t=t+k;
72
73     if video_on && (mod(freq,10)==0)
74         plot(x,V,'r','LineWidth',1);
75         % plot(x,U_exact(x,t),'b','LineWidth',1);
76         title(['Numerical solution at t = ',num2str(t)]);
77         axis(theAxes);
78         grid; xlabel('x');
79         legend('v')

```



```

80         ax = gca;
81         ax.FontSize = 16;
82         currFrame = getframe;
83         writeVideo(vidObj,currFrame);
84     end
85     freq = freq + 1;
86 end
87
88 if video_on
89     close(vidObj);
90 end
91
92 U = U_exact(x,t,eps);
93 error = sqrt((U-V)'*H*(U-V))
94
95
96 function u_exact = U_exact(x,t,eps)
97 c = 2;
98 a = 1;
99
100 u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
101 end

```

## A.2 Source code Task 6

```

1  video_on = 1;
2
3
4  m = 51;
5  x_l = -1 ; x_r = 1;
6  len = x_r - x_l;
7  eps= 1e-6;
8  t_end = 0.4;
9
10 h=(x_r-x_l)/(m-1);
11 if video_on
12     theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
13     scrsz = get(0, 'ScreenSize');
14     figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
15     vidObj = VideoWriter('System');
16     open(vidObj);
17 end
18
19 CFL=0.5;
20 k=CFL*h^(2);
21
22 c= ones(m,1);
23
24
25 % Uncomment one of these
26 %{
27 SBP6_Variable
28 gamma = 0.17;
29 DI = -gamma*HI*DD_3'*DD_3;
30 %}
31
32 %{
33 SBP4_Variable
34 gamma = 0.4;
35 DI = -gamma*HI*DD_2'*DD_2;
36 %}
37
38
39 SBP2_Variable

```

```

40 gamma = 0.5;
41 DI = -gamma*HI*DD_1'*DD_1;
42
43 I = eye(m);
44 eps_bar = I*eps;
45
46
47 L = [ e_1'; e_m'];
48 P=I-HI*L'*((L*HI*L')\L);
49 g = zeros(m,1);
50 PB = HI*L'*((L*HI*L')\L);
51
52 t=0;
53 x=linspace(x_l,x_r,m)';
54 V=zeros(m,1);
55 V = U_exact(x,0,eps);
56 freq = 0;
57
58 max_itter=floor(t_end/k);
59 for nr_itter=1:max_itter
60
61
62
63
64     g(1) = U_exact(x_l,t,eps);
65     g(m) = U_exact(x_r,t,eps);
66     V0 = P*V+PB*g;
67
68     w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
69         + P*eps_bar*D2*V0 + DI*V0 ;
70     %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g)) ;% + Dp*(eps_bar)*Dm*V;
71
72     g(1) = U_exact(x_l,t+k/2,eps);
73     g(m) = U_exact(x_r,t+k/2,eps);
74     V1 = P*(V+k/2*w1) +PB*g;
75
76     w2=-1/3*P*D1*(diag(V1)*V1) -1/3*P*diag(V1)*D1*V1 ...
77         + P*eps_bar*D2*V1 +DI*V1;
78     %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;% + Dp*(eps_bar)*Dm*V1;
79
80     g(1) = U_exact(x_l,t+k/2,eps);
81     g(m) = U_exact(x_r,t+k/2,eps);
82     V2 = P*(V+k/2*w2)+PB*g;
83
84     w3=-1/3*P*D1*(diag(V2)*V2) -1/3*P*diag(V2)*D1*V2 ...
85         + P*eps_bar*D2*V2 +DI*V2;
86     %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g)) ;% + Dp*(eps_bar)*Dm*V2;
87
88     g(1) = U_exact(x_l,t+k,eps);
89     g(m) = U_exact(x_r,t+k,eps);
90     V3 = P*(V+k*w3)+PB*g;
91     w4=-1/3*P*D1*(diag(V3)*V3) -1/3*P*diag(V3)*D1*V3 ...
92         + P*eps_bar*D2*V3 +DI*V3;
93     %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g)) ;% + Dp*(eps_bar)*Dm*V3;
94     V=V+k/6*(w1+2*w2+2*w3+w4);
95
96     t=t+k;
97
98     if video_on && (mod(freq,2)==0)
99
100         plot(x,V,'r','LineWidth',1);
101
102         % plot(x,U_exact(x,t),'r','LineWidth',1);
103         title(['Numerical solution at t = ',num2str(t)]);
104         axis(theAxes);
105         grid; xlabel('x');
106         legend('v')

```

```

107         ax = gca;           % current axes
108         ax.FontSize = 16;
109         currFrame = getframe;
110         writeVideo(vidObj,currFrame);
111     end
112     freq = freq + 1;
113 end
114
115 if video_on
116     close(vidObj);
117 end
118
119 U = U_exact(x,t,eps);
120 error = sqrt((U-V)'*H*(U-V))
121
122
123 function u_exact = U_exact(x,t,eps)
124 c = 2;
125 a = 1;
126 u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
127 end

```

### A.3 Source code Task 7

```

1  video_on = 1;
2
3
4  m = 51;
5  x_l = -1 ; x_r = 1;
6  len = x_r - x_l;
7  eps= 1e-6;
8  t_end = 0.4;
9
10 h=(x_r-x_l)/(m-1);
11 if video_on
12     theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
13     scrsz = get(0, 'ScreenSize');
14     figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
15     vidObj = VideoWriter('System');
16     open(vidObj);
17 end
18
19 CFL=0.5;
20 k=CFL*h^(2);
21
22 c=ones(m,1);
23
24
25 %Uncomment one of these:
26 %{
27 SBP6_Variable
28 gamma = 0.17;
29 DI = -gamma*HI*DD_3'*DD_3;
30 %}
31
32 %{
33 SBP4_Variable
34 gamma = 0.4;
35 DI = -gamma*HI*DD_2'*DD_2;
36 %}
37
38
39 SBP2_Variable
40 gamma = 0.5;

```

```

41 DI = -gamma*HI*DD_1'*DD_1;
42
43 I = eye(m);
44 eps_bar = I*eps;
45
46
47 %L = [ e_1'; e_m'];
48 %P=I-HI*L'*((L*HI*L')\L);
49 g = zeros(2,1);
50 %PB = HI*L'*((L*HI*L')\L);
51
52 t=0;
53 x=linspace(x_l,x_r,m)';
54 V=zeros(m,1);
55 V = U_exact(x,0,eps);
56 freq = 0;
57
58 max_itter=floor(t_end/k);
59 for nr_itter=1:max_itter
60
61     L = [1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1;
62          1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m];
63     P=I-HI*L'*((L*HI*L')\L);
64     %PB = HI*L'*((L*HI*L')\L);
65     %UU = U_exact(x,t,eps);
66     var1 = U_exact(x_l,t,eps); var2 = U_exact(x_r,t,eps);
67     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t,eps);
68     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t,eps);
69     PB = HI*L'*((L*HI*L')\g);
70     V0 = (P)*V+PB;
71
72     w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
73         + P*eps_bar*D2*V0 + DI*V0 ;
74     %w1=-0.5*P*D1*(diag(P*V+PB*g)*(P*V+PB*g)) ;% + Dp*(eps_bar)*Dm*V;
75
76     UU = U_exact(x,t+k/2,eps);
77     var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
78     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t+k/2,eps);
79     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t+k/2,eps);
80     PB = HI*L'*((L*HI*L')\g);
81     V1 = (P)*(V+k/2*w1)+PB ;
82
83     w2=-1/3*P*D1*(diag(V1)*V1) -1/3*P*diag(V1)*D1*V1 ...
84         + P*eps_bar*D2*V1 +DI*V1;
85     %w2=-0.5*P*D1*(diag(P*V1+PB*g)*(P*V1+PB*g)) ;% + Dp*(eps_bar)*Dm*V1;
86
87     var1 = U_exact(x_l,t+k,eps); var2 = U_exact(x_r,t+k,eps);
88     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k,eps);
89     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k,eps);
90     PB = HI*L'*((L*HI*L')\g);
91     V2 = (P)*(V+k/2*w2)+PB ;
92
93     w3=-1/3*P*D1*(diag(V2)*V2) -1/3*P*diag(V2)*D1*V2 ...
94         + P*eps_bar*D2*V2 +DI*V2;
95     %w3=-0.5*P*D1*(diag(P*V2+PB*g)*(P*V2+PB*g)) ;% + Dp*(eps_bar)*Dm*V2;
96     UU = U_exact(x,t+k,eps);
97     var1 = U_exact(x_l,t+k,eps); var2 = U_exact(x_r,t+k,eps);
98     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k,eps);
99     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k,eps);
100    PB = HI*L'*((L*HI*L')\g);
101    V3 = (P)*(V+k*w3)+PB;
102    w4=-1/3*P*D1*(diag(V3)*V3) -1/3*P*diag(V3)*D1*V3 ...
103        + P*eps_bar*D2*V3 +DI*V3;
104    %w4=-0.5*P*D1*(diag(P*V3+PB*g)*(P*V3+PB*g)) ;% + Dp*(eps_bar)*Dm*V3;
105    V=V+k/6*(w1+2*w2+2*w3+w4);
106
107    t=t+k;

```

```

108
109     if video_on && (mod(freq,2)==0)
110
111         plot(x,V,'r','LineWidth',1);
112
113         % plot(x,U_exact(x,t,eps),'r','LineWidth',1);
114
115         % plot(x,U_diff_exact(x,t,eps),'b','LineWidth',1);
116         title(['Numerical solution at t = ',num2str(t)]);
117         axis(theAxes);
118         grid; xlabel('x');
119         legend('v')
120         ax = gca; % current axes
121         ax.FontSize = 16;
122         currFrame = getframe;
123         writeVideo(vidObj,currFrame);
124     end
125     freq = freq + 1;
126 end
127
128 if video_on
129     close(vidObj);
130 end
131
132 U = U_exact(x,t,eps);
133 error = sqrt((U-V)'*H*(U-V))
134
135
136 function u_exact = U_exact(x,t,eps)
137 c = 2;
138 a = 1;
139 u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
140 end
141
142 function u_diff_exact = U_diff_exact(x,t,eps)
143 u_diff_exact = -(tanh((2*t - x)/(2*eps)).^2 - 1)/(2*eps);
144 end

```

## A.4 Source code Task 8

```

1 % BDF1
2 % This in the file BDF.m
3 function bdf = BDF(VV,command)
4 if command == 0
5     bdf = 0*VV(7,:);
6 end
7 if command == 1
8     bdf = VV(7,:) - VV(6,:);
9 end
10
11 if command == 2
12     bdf = VV(7,:) -(4/3)*VV(6,:)+(1/3)*VV(5,:);
13 end
14
15 if command == 3
16     bdf = VV(7,:) -(18/11)*VV(6,:)+(9/11)*VV(5,:) - ...
17         (2/11)*VV(4,:);
18 end
19
20 if command == 4
21     bdf = VV(7,:) -(48/25)*VV(6,:)+(36/25)*VV(5,:) - ...
22         (16/25)*VV(4,:)+(3/25)*VV(3,:);
23 end
24

```

```

25 if command == 5
26     bdf = VV(7,:) -(300/137)*VV(6,:)+(300.137)*VV(5,:) - ...
27         (200/137)*VV(4,:)+(75/137)*VV(3,:)-(12/137)*VV(2,:);
28 end
29
30 if command >= 6
31     bdf = VV(7,:) -(360/147)*VV(6,:)+(450/147)*VV(5,:) -...
32         (400/147)*VV(4,:)+(225/147)*VV(3,:)-(72/147)*VV(2,:)+...
33         (10/147)*VV(1,:);
34 end
35
36 end
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 % in the file Task8.m
40 video_on = 1;
41
42
43 m = 51;
44 x_l = -1 ; x_r = 1;
45 len = x_r - x_l;
46 eps = 1e-6;
47 t_end = 0.4;
48
49 h=(x_r-x_l)/(m-1);
50 if video_on
51     theAxes=[x_l x_r -0.5 3.5]; % Regarding the figure
52     scrsz = get(0,'ScreenSize');
53     figure('Position',[scrsz(3)/2 scrsz(4) scrsz(3)/2 scrsz(4)])
54     vidObj = VideoWriter('System');
55     open(vidObj);
56 end
57
58 CFL=0.5;
59 k=CFL*h^(2);
60
61
62
63 c=ones(m,1);
64 SBP4_Variable;
65
66 I = eye(m);
67 eps_bar = I*eps;
68
69
70 %L = [ e_l'; e_m'];
71 %P=I-HI*L'*((L*HI*L')\L);
72 g = zeros(2,1);
73 %PB = HI*L'*((L*HI*L')\L);
74
75 t=0;
76 x=linspace(x_l,x_r,m)';
77 V=zeros(m,1);
78 V = U_exact(x,0,eps);
79 freq = 0;
80
81 % store the last 6 solution
82 VV_store = [0*V';0*V';0*V';0*V';0*V';0*V';V'];
83
84 % BDF_command tells which order PDF to use
85 BDF_command = 0;
86 Residual = zeros(m,1);
87 BDF_u = zeros(1,m);
88
89
90
91 max_itter=floor(t_end/k);

```

```

92 for nr_iter=1:max_iter
93
94     % BDF time integral for the viscosity
95     BDF_u=BDF(VV_store,BDF_command);
96
97
98     L = [1/6*(V(1)+abs(V(1)))*e_1'-eps*d_1;
99           1/6*(V(m)-abs(V(m)))*e_m'-eps*d_m];
100     P=I-HI*L'*((L*HI*L')\L);
101
102     % related to RK4 stage1
103     var1 = U_exact(x_l,t,eps); var2 = U_exact(x_r,t,eps);
104     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t,eps);
105     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t,eps);
106     PB = HI*L'*((L*HI*L')\g);
107     V0 = (P)*V +PB;
108
109     % Compute viscosity
110     % c=ones(m,1);
111     %D2=HI*(-M-diag(c)*e_1*d_1+diag(c)*e_m*d_m);
112     Residual = BDF_u' +1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
113               - P*D2*V0;
114
115     % we dont use local for start
116
117     eps_r = h^(2)* abs(Residual) ./ ( abs(V0-max(V0-mean(V0))) );
118     eps_l = 0.5*h *abs(0.5*V0);
119     %eps_viscosity
120     c = min(eps_l,eps_r);
121     SBP4_Variable;
122
123
124     %%%% RK4 starts here
125     % Stage 1
126     w1=-1/3*P*D1*(diag(V0)*V0) -1/3*P*diag(V0)*D1*V0 ...
127         + P*D2*V0 ;
128
129
130     % Stage 2
131     var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
132     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_l,t+k/2,eps);
133     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_r,t+k/2,eps);
134     PB = HI*L'*((L*HI*L')\g);
135     V1 = (P)*(V+k/2*w1)+PB ;
136
137     w2=-1/3*P*D1*(diag(V1)*V1) -1/3*P*diag(V1)*D1*V1 ...
138         + P*D2*V1 ;
139
140     % Stage 3
141     var1 = U_exact(x_l,t+k/2,eps); var2 = U_exact(x_r,t+k/2,eps);
142     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k/2,eps);
143     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k/2,eps);
144     PB = HI*L'*((L*HI*L')\g);
145     V2 = (P)*(V+k/2*w2)+PB ;
146
147     w3=-1/3*P*D1*(diag(V2)*V2) -1/3*P*diag(V2)*D1*V2 ...
148         + P*D2*V2 ;
149
150     % Stage4
151     var1 = U_exact(x_l,t+k,eps); var2 = U_exact(x_r,t+k,eps);
152     g(1) = 1/6*(var1+abs(var1))*var1 - eps*U_diff_exact(x_r,t+k,eps);
153     g(2) = 1/6*(var2-abs(var2))*var2 - eps*U_diff_exact(x_l,t+k,eps);
154     PB = HI*L'*((L*HI*L')\g);
155     V3 = (P)*(V+k*w3) +PB;
156     w4=-1/3*P*D1*(diag(V3)*V3) -1/3*P*diag(V3)*D1*V3 ...
157         + P*D2*V3 ;
158

```

```

159     V=V+k/6*(w1+2*w2+2*w3+w4);
160
161     t=t+k;
162
163     if video_on && (mod(freq,2)==0)
164
165         plot(x,V,'r','LineWidth',1);
166
167         % plot(x,U_exact(x,t,eps),'r','LineWidth',1);
168
169         % plot(x,U_diff_exact(x,t,eps),'b','LineWidth',1);
170         title(['Numerical solution at t = ',num2str(t)]);
171         axis(theAxes);
172         grid; xlabel('x');
173         legend('v')
174         ax = gca; % current axes
175         ax.FontSize = 16;
176         currFrame = getframe;
177         writeVideo(vidObj,currFrame);
178     end
179     freq = freq + 1;
180
181     %shift the stored solution to the left and add the new one
182     VV_store = circshift(VV_store,-1);
183     VV_store(7,:) = V';
184     BDF_command = BDF_command +1;
185
186 end
187
188 if video_on
189     close(vidObj);
190 end
191
192 U = U_exact(x,t,eps);
193 error = sqrt((U-V)'*H*(U-V))
194
195
196 function u_exact = U_exact(x,t,eps)
197 c = 2;
198 a = 1;
199 u_exact = c-a*tanh(a*(x-c*t)/(2*eps));
200 end
201
202 function u_diff_exact = U_diff_exact(x,t,eps)
203 u_diff_exact = -(tanh((2*t - x)/(2*eps)).^2 - 1)/(2*eps);
204 end

```

## A.5 Source code Part 2.2

### A.5.1 GFEM

```

1 from dolfin import*
2 import numpy as np
3 import scipy.linalg as la
4 import ufl
5 import sympy as symp
6
7
8
9 T = 2
10 num_samples = 40
11 dt = 0.05
12
13

```



```

14 mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
15 V = FunctionSpace(mesh, "CG", 1)
16 VV = VectorFunctionSpace(mesh,"CG",1)
17
18 def boundary(x, on_boundary):
19     return on_boundary
20
21
22 bc = DirichletBC(V,Constant(0.0),boundary)
23
24 ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2,pi=np.pi)
25
26
27 u0 = interpolate(ID, V)
28
29 u = TrialFunction(V)
30 v = TestFunction(V)
31 #B = Expression(('sin(u)','cos(u)'),degree=2, u=u)
32 #B0 = Expression(('sin(u)','cos(u)'), degree=2,u=u0)
33
34 def B(u):
35     # return Expression(('cos(u)', '-sin(u)'),degree=2,u=u)
36     return as_vector((cos(u), -sin(u)))
37
38
39 u_k = interpolate(Constant(0.0),V)
40 eps=1.0
41 tol = 1.0E-5
42 maxiter=25
43
44
45
46 a = u*v*dx + 0.5*dt*dot(B(u_k),grad(u)) *v*dx
47 L = +u0*v*dx - 0.5*dt*dot(B(u0),grad(u0)) *v*dx
48
49 #a = u*v*dx + 0.5*dt*div(B(u))
50 u = Function(V)
51
52 out_file = File("VTK/Results.pvd", "compressed")
53
54
55
56 #u.assign(u0)
57 t=0
58 t_save =0.0
59 out_file << (u0,t)
60
61 while t<=T:
62     t += dt
63     t_save += dt
64
65     itera = 0
66     eps =np.Inf
67
68     while eps>tol and itera <maxiter:
69         itera += 1
70
71         solve(a==L,u,bc)
72         diff = np.array(u.vector()) - np.array(u_k.vector())
73
74         eps = np.linalg.norm(diff, ord=np.Inf)
75         u_k.assign(u)
76
77     u0.assign(u)
78
79     if t_save >T/num_samples or t>=T-dt:
80         print('time = ',t)

```

```

81         out_file << (u,t)
82         t_save =0

```

### A.5.2 GLS

```

1  from dolfin import*
2  import numpy as np
3  import scipy.linalg as la
4  import ufl
5
6
7
8  T = 2
9  num_samples = 40
10 dt = 0.05
11
12
13 mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
14 V = FunctionSpace(mesh, "CG", 1)
15 VV = VectorFunctionSpace(mesh, "CG",1)
16
17 def boundary(x, on_boundary):
18     return on_boundary
19
20
21 bc = DirichletBC(V, Constant(0.0), boundary)
22
23 ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2,pi=np.pi)
24
25
26 u0 = interpolate(ID, V)
27
28 u = TrialFunction(V)
29 v = TestFunction(V)
30 #B = Expression(('sin(u)', 'cos(u)'), degree=2, u=u)
31 #B0 = Expression(('sin(u)', 'cos(u)'), degree=2, u=u0)
32
33 def B(u):
34     # return project(Expression(('cos(u)', 'sin(u)'), degree=2, u=u), VV)
35     return as_vector((cos(u), -sin(u)))
36     # return np.array([sym(cos(u)), sym(sin(u))])
37
38
39 #def f(u):
40     # return as_vector((sin(u), cos(u)))
41
42 u_k = interpolate(Constant(0.0), V)
43 eps=1.0
44 tol = 1.0E-5
45 maxiter=25
46
47 h = 4/31
48 gamma = 0.5*h
49
50 a = u*v*dx + 0.5*dt*dot(B(u_k), grad(u)) *v*dx \
51     +gamma*u*dot(B(u0), grad(v))*dx \
52     +0.5*gamma*dt*dot(B(u_k), grad(u))*dot(B(u0), grad(v))*dx
53
54 L = u0*v*dx - 0.5*dt*dot(B(u0), grad(u0)) *v*dx \
55     +gamma*u0*dot(B(u0), grad(v))*dx \
56     -0.5*gamma*dt*dot(B(u0), grad(u0))*dot(B(u0), grad(v))*dx
57
58 u = Function(V)
59

```

```

60
61
62 out_file = File("VTK/Results.pvd", "compressed")
63
64
65 #u.assign(u0)
66 t=0
67 t_save =0.0
68 out_file << (u0,t)
69
70 while t<=T:
71     t += dt
72     t_save += dt
73
74     itera = 0
75     eps =np.Inf
76
77     while eps>tol and itera <maxiter:
78         itera += 1
79
80         solve(a=L,u,bc)
81         diff = np.array(u.vector()) - np.array(u_k.vector())
82
83         eps = np.linalg.norm(diff, ord=np.Inf)
84         u_k.assign(u)
85
86     u0.assign(u)
87
88     if t_save >T/num_samples or t>=T-dt:
89         print('time = ',t)
90         out_file << (u,t)
91         t_save =0

```

### A.5.3 RV

```

1 from dolfin import*
2 import numpy as np
3 import scipy.linalg as la
4 import ufl
5
6
7
8 T = 2.0
9 num_samples = 40
10 dt = 0.05
11
12
13 mesh = RectangleMesh(Point(-2,-2.5),Point(2,1.5),31,31)
14 V = FunctionSpace(mesh, "CG", 1)
15 VV = VectorFunctionSpace(mesh, "CG", 1)
16
17 def boundary(x, on_boundary):
18     return on_boundary
19
20
21 bc = DirichletBC(V, Constant(0.0), boundary)
22
23 ID = Expression('pow(pow(x[0],2)+pow(x[1],2),0.5) <=1?14/4*pi:pi/4', degree=2, pi=np.pi)
24
25
26 u0 = interpolate(ID, V)
27
28 u = TrialFunction(V)
29 v = TestFunction(V)

```

```

30 #B = Expression(('sin(u)', 'cos(u)'), degree=2, u=u)
31 #B0 = Expression(('sin(u)', 'cos(u)'), degree=2, u=u0)
32
33 def B(u):
34     # return project(Expression(('cos(u)', '-sin(u)'), degree=2, u=u), VV)
35     return as_vector((cos(u), -sin(u)))
36     # return np.array([sym(cos(u)), sym(-sin(u))])
37
38
39 #def f(u):
40     # return as_vector((sin(u), cos(u)))
41
42 u_k = interpolate(Constant(0.0), V)
43 eps=1.0
44 tol = 1.0E-5
45 maxiter=25
46
47 #h = CellDiameter(mesh)
48 h = 4/30
49 EPS = interpolate(Constant(0.0), V)
50
51 """
52 def max_norm(u):
53     u_array = np.array(u.vector())
54     Max = np.max(u_array - np.mean(u_array))
55     # MAX = np.max(u_array - assemble(u*dx)/assemble(1*dx))
56     return Max
57 """
58
59
60 a = u*v*dx + 0.5*dt*dot(B(u_k), grad(u)) *v*dx \
61     +0.5*dt*dot(EPS*grad(u), grad(v))*dx
62
63
64 L = u0*v*dx - 0.5*dt*dot(B(u0), grad(u0)) *v*dx \
65     - 0.5*dt*dot(EPS*grad(u0), grad(v))*dx
66
67 u = Function(V)
68
69
70
71
72 out_file = File("VTK/Results.pvd", "compressed")
73
74
75 #u.assign(u0)
76 t=0
77 t_save =0.0
78 out_file << (u0, t)
79
80 while t<=T:
81     t += dt
82     t_save += dt
83
84     itera = 0
85     eps =np. Inf
86
87     while eps>tol and itera <maxiter:
88         itera += 1
89
90         solve(a==L, u, bc)
91         diff = np.array(u.vector()) - np.array(u_k.vector())
92
93         eps = np.linalg.norm(diff, ord=np. Inf)
94         u_k.assign(u)
95
96     u_array = np.array(u.vector())

```

```

97     u0_array = np.array(u0.vector())
98     flux = project(dot(B(u), grad(u)), V)
99     flux_array = np.array(flux.vector())
100
101     Res = dt*(u_array - u0_array) + flux_array
102     beta = norm(project(B(u), VV).vector(), 'linf')
103
104     dX = Measure('dx', mesh)
105     max_normm = np.max(u_array - assemble(u*dX)/assemble(1*dX))
106     epsilons = np.minimum(h*beta, 0.25*h**2 *abs(Res)/max_normm)
107
108     EPS.vector()[:] = epsilons
109     # EPS.assign(epsilons)
110
111
112     u0.assign(u)
113
114     if t_save > T/num_samples or t >= T-dt:
115         print('time = ', t)
116         out_file << (u, t)
117         t_save = 0

```