


Name	Ahmed Saeed Mohamed Sherif
Course	Finite Elements in Fluids
Master	MSc Computational Mechanics
Report no.	4
Topics	1D unsteady convection (Cosine profile - Steep front) 1D unsteady convection-diffusion (Gaussian hill)

Contents


Contents	i
1 1D unsteady convection	1
1.1 Propagation of a cosine profile	1
1.1.1 Explicit methods	1
1.1.2 Implicit methods	2
1.2 Propagation of a steep front	4
1.2.1 Crank-Nicholson with linear finite elements for Galerkin formulation . . .	4
1.2.2 Crank-Nicholson with linear finite elements for least-squares formulation	4
1.2.3 Lax-Wendroff with linear finite elements for Galerkin formulation	8
1.2.4 Two-step Lax-Wendroff with linear finite elements for Galerkin formulation	9
2 1D unsteady convection-diffusion	12
2.1 Galerkin and Crank-Nicholson	12
2.2 Galerkin and $R_{2,2}$	13
2.3 Galerkin and $R_{3,3}$	13
2.4 Galerkin and Adams-Bashforth	14
2.5 Time-discontinuous Galerkin formulation for the convection-diffusion equation .	17
A Developed codes for 1D unsteady convection	19
A.1 New function <code>system_CN_LS.m</code>	19
A.2 New function <code>system_LW_2S.m</code>	20
B Developed codes for 1D unsteady convection-diffusion	22
B.1 Modified function <code>Galerkin.m</code>	22
References	24

1 1D unsteady convection

1.1 Propagation of a cosine profile

$$\begin{cases} u_t + au_x = 0 & x \in (0, 1), t \in (0, 0.6] \\ u(x, 0) = u_0(x) & x \in (0, 1) \\ u(0, t) = 0 & t \in (0, 0.6] \end{cases}$$

$$u_0(x) = \begin{cases} \frac{1}{2}(1 + \cos(\pi(x - x_0)/\sigma)) & \text{if } |x - x_0| \leq \sigma, \\ 0 & \text{otherwise} \end{cases}$$

$a = 1, x_0 = 0.2, \sigma = 0.12, \Delta x = 2 \cdot 10^{-2}$



In all the following cases, linear finite elements of Galerkin spatial discretization is used, while different time-integration methods are tested and compared.

1.1.1 Explicit methods

1. Second-order Lax-Wendroff with consistent mass matrix (TG2)
2. Second-order Lax-Wendroff with lumped mass matrix (LW-FD)
3. Third-order Taylor-Galerkin (TG3)

Explicit methods results:

1. The results obtained using the different explicit methods at various values of the Courant number, C , are shown in Figure 1.
2. For Lax-Wendroff with consistent mass matrix (TG2), it is observed that the method is unstable at $C = 0.75$, which agrees with the theory, as the stability limit for this method is $C < \frac{1}{3}$.
3. To increase the stability limit for (TG2), a lumped mass matrix is used, but this comes at the cost of accuracy. It is observed that the Lax-Wendroff method with lumped mass matrix (LW-DF) has lower phase accuracy when compared to the other two methods.
4. As for the third-order Taylor-Galerkin method (TG3), it is observed that the phase accuracy is better, and it is uniform over the entire stable interval $0 < C < 1$.

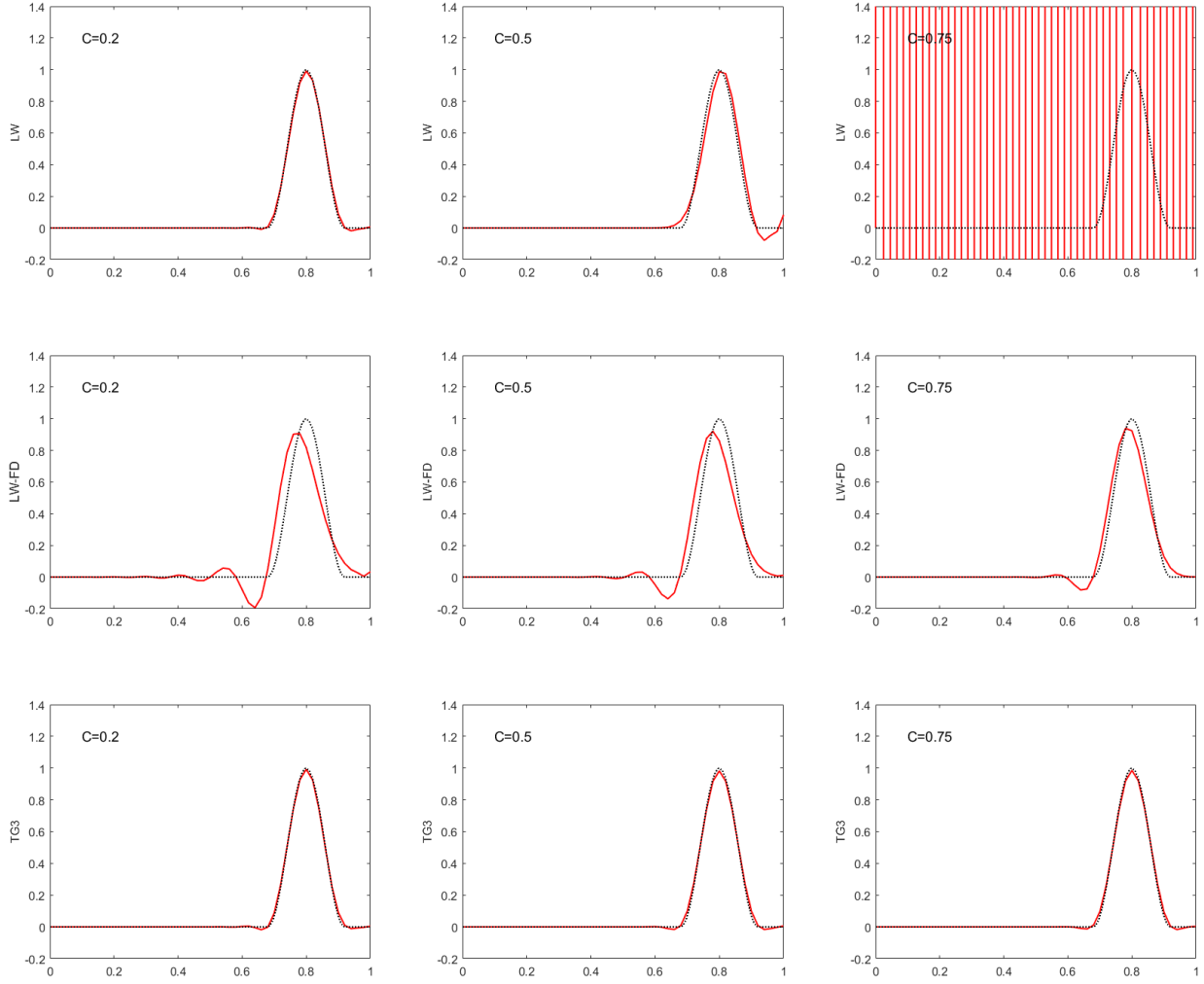


Figure 1: Propagation of a cosine profile using explicit methods, (Top): TG2, (Middle): LW-FD, (Bottom): TG3

1.1.2 Implicit methods

1. Second-order Crank-Nicholson with consistent mass matrix (CN-FE)
2. Second-order Crank-Nicholson with lumped mass matrix (CN-FD)
3. Implicit fourth-order Taylor-Galerkin with consistent mass matrix (TG4)

Implicit methods results:

1. The results obtained using the different implicit methods at various values of the Courant number, C , are shown in Figure 2.
2. Crank-Nicholson method is unconditionally stable. However, lower accuracy is observed at higher values of C .

3. Again, using lumped mass matrix leads to degradation of the phase accuracy as seen for Crank-Nicholson with lumped mass matrix (CN-FD).
4. As for the implicit fourth-order Taylor-Galerkin method (TG4), it is showing excellent phase accuracy at all values of C .

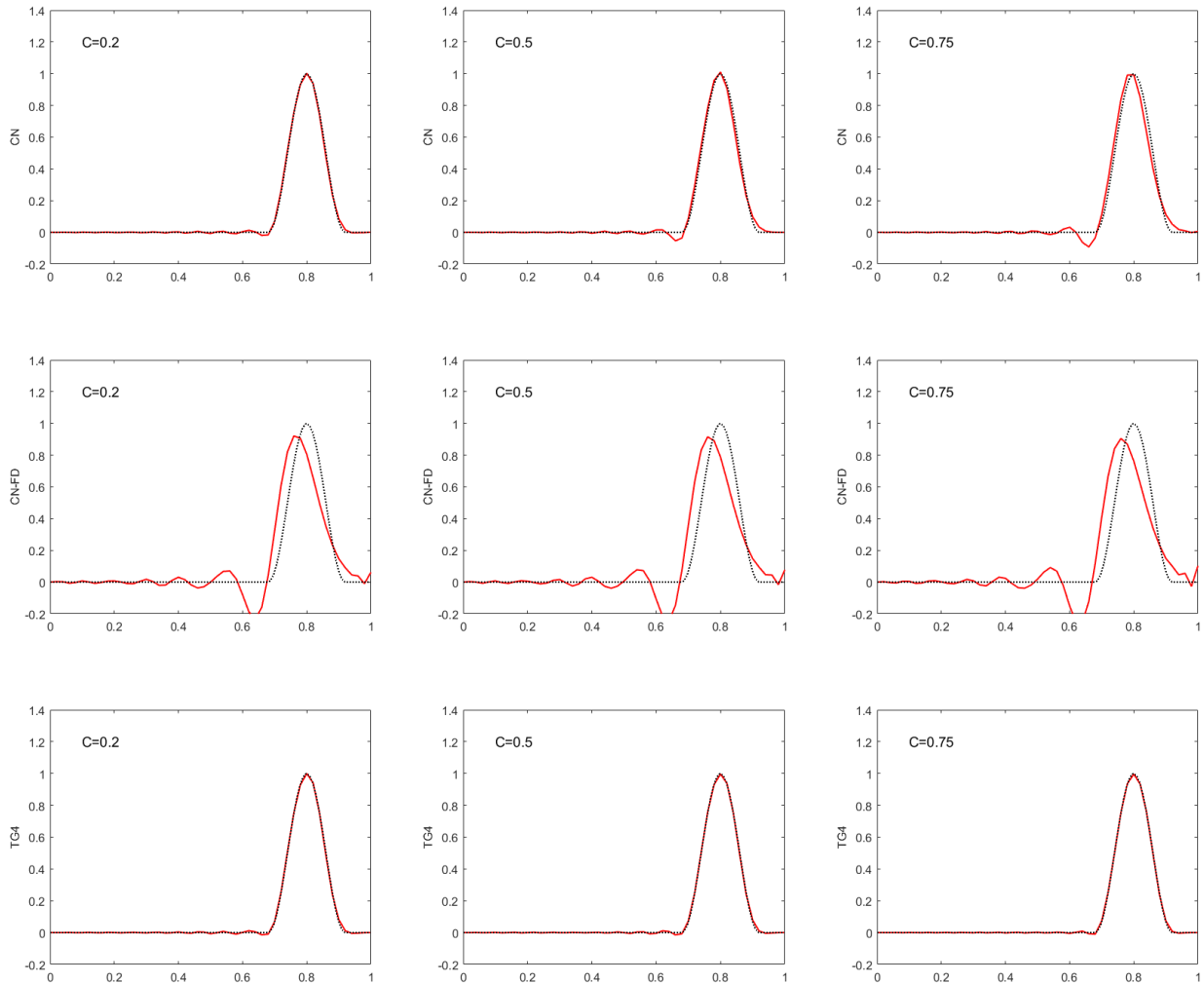


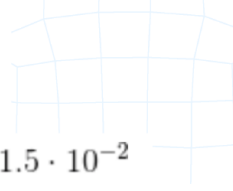
Figure 2: Propagation of a cosine profile using implicit methods, (Top): CN-FE, (Middle): CN-FD, (Bottom): TG4

1.2 Propagation of a steep front

$$\begin{cases} u_t + au_x = 0 & x \in (0, 1), t \in (0, 0.6] \\ u(x, 0) = u_0(x) & x \in (0, 1) \\ u(0, t) = 1 & t \in (0, 0.6] \end{cases}$$

$$u_0(x) = \begin{cases} 1 & \text{if } x \leq 0.2, \\ 0 & \text{otherwise} \end{cases}$$

$$a = 1, \Delta x = 2 \cdot 10^{-2}, \Delta t = 1.5 \cdot 10^{-2}$$



Under those conditions, the Courant–Friedrichs–Lewy (CFL) condition is:

$$C = \frac{a\Delta t}{\Delta x} = \frac{1 * 0.015}{0.02} = 0.75$$

1.2.1 Crank-Nicholson with linear finite elements for Galerkin formulation

Employing Crank-Nicholson time integration with linear finite elements for the Galerkin spatial discretization, the obtained solution is shown in Figure 3. It is noted that even though the method is unconditionally stable, spurious oscillations appear due to the lack of artificial diffusion, which results in a non-accurate solution.

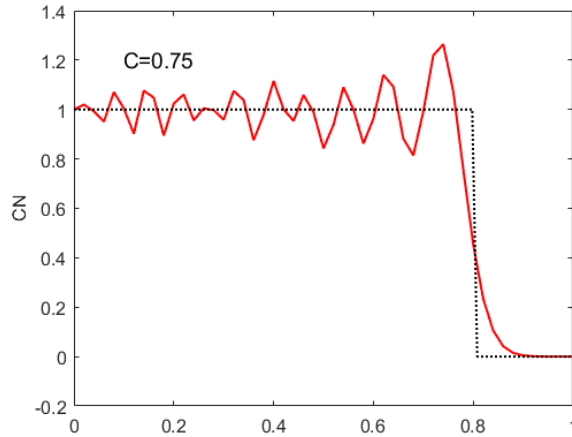


Figure 3: Propagation of steep front using the Crank-Nicholson scheme with the Galerkin method.

1.2.2 Crank-Nicholson with linear finite elements for least-squares formulation

Considering the linear convection equation with source term, the Crank-Nicholson time discretization is given by:

$$\frac{\Delta u}{\Delta t} + \frac{1}{2}(\mathbf{a} \cdot \nabla)\Delta u = \frac{1}{2}s^{n+1} + \frac{1}{2}s^n - \mathbf{a} \cdot \nabla u^n \quad (1)$$

which can be viewed as:

$$\mathcal{L}(\Delta u) - f = 0 \quad (2)$$

where $\mathcal{L} = \frac{1}{\Delta t} + \frac{1}{2}\mathbf{a} \cdot \nabla$, and $f = \frac{1}{2}s^{n+1} + \frac{1}{2}s^n - \mathbf{a} \cdot \nabla u^n$

the least-square weak formulation is written as:

$$(\mathcal{L}(w), \mathcal{L}(\Delta u) - f)_\Omega = 0 \quad (3)$$

which yields,

$$\left(\frac{w}{\Delta t} + \frac{1}{2}\mathbf{a} \cdot \nabla w, \frac{\Delta u}{\Delta t} + \frac{1}{2}\mathbf{a} \cdot \nabla \Delta u\right)_\Omega = \left(\frac{w}{\Delta t} + \frac{1}{2}\mathbf{a} \cdot \nabla w, \frac{1}{2}s^{n+1} + \frac{1}{2}s^n - \mathbf{a} \cdot \nabla u^n\right)_\Omega \quad (4)$$

further manipulation,

$$\begin{aligned} & \frac{1}{\Delta t^2}(w, \Delta u)_\Omega + \frac{1}{2\Delta t}(w, \mathbf{a} \cdot \nabla \Delta u)_\Omega + \frac{1}{2\Delta t}(\mathbf{a} \cdot \nabla w, \Delta u)_\Omega + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega \\ &= \frac{1}{2\Delta t}(w, s^{n+1})_\Omega + \frac{1}{2\Delta t}(w, s^n)_\Omega - \frac{1}{\Delta t}(w, \mathbf{a} \cdot \nabla u^n)_\Omega + \frac{1}{4}(\mathbf{a} \cdot \nabla w, s^{n+1})_\Omega \\ & \quad + \frac{1}{4}(\mathbf{a} \cdot \nabla w, s^n)_\Omega - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega \end{aligned} \quad (5)$$

assuming that the source term is not time-dependent ($s^{n+1} = s^n = s$) yields,

$$\begin{aligned} & \frac{1}{\Delta t^2}(w, \Delta u)_\Omega + \frac{1}{2\Delta t}(w, \mathbf{a} \cdot \nabla \Delta u)_\Omega + \frac{1}{2\Delta t}(\mathbf{a} \cdot \nabla w, \Delta u)_\Omega + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega \\ &= -\frac{1}{\Delta t}(w, \mathbf{a} \cdot \nabla u^n)_\Omega - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega + \frac{1}{\Delta t}(w, s)_\Omega + \frac{1}{2}(\mathbf{a} \cdot \nabla w, s)_\Omega \end{aligned} \quad (6)$$

integrating by parts the second term on LHS and the first term on RHS yields:

$$\begin{aligned} & \frac{1}{\Delta t^2}(w, \Delta u)_\Omega - \frac{1}{2\Delta t}(\mathbf{a} \cdot \nabla w, \Delta u)_\Omega + \frac{1}{2\Delta t}((\mathbf{a} \cdot \mathbf{n})w, \Delta u)_\Gamma \\ & \quad + \frac{1}{2\Delta t}(\mathbf{a} \cdot \nabla w, \Delta u)_\Omega + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega \\ &= \frac{1}{\Delta t}(\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{1}{\Delta t}((\mathbf{a} \cdot \mathbf{n})w, u^n)_\Gamma - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega \\ & \quad + \frac{1}{\Delta t}(w, s)_\Omega + \frac{1}{2}(\mathbf{a} \cdot \nabla w, s)_\Omega \end{aligned} \quad (7)$$

given that the boundary $\Gamma = \Gamma^{in} \cup \Gamma^{out}$, and recalling that for hyperbolic problems the boundary conditions are applied only to inflow boundaries. Furthermore, the inflow boundary could be a Neumann part and a Dirchlet part, i.e. $\Gamma^{in} = \Gamma_N^{in} \cup \Gamma_D^{in}$. Since $w = 0$ on the Dirchlet boundary, therefore the weak form is written as:

$$\begin{aligned} & \frac{1}{\Delta t^2}(w, \Delta u)_\Omega + \frac{1}{2\Delta t}((\mathbf{a} \cdot \mathbf{n})w, \Delta u)_{\Gamma^{out}} + \frac{1}{2\Delta t}((\mathbf{a} \cdot \mathbf{n})w, \Delta u)_{\Gamma_N^{in}} + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega \\ &= \frac{1}{\Delta t}(\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{1}{\Delta t}((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma^{out}} - \frac{1}{\Delta t}((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma_N^{in}} - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega \\ & \quad + \frac{1}{\Delta t}(w, s)_\Omega + \frac{1}{2}(\mathbf{a} \cdot \nabla w, s)_\Omega \end{aligned} \quad (8)$$

particularizing it to the problem at hand (no Neumann boundary) yields,

$$\begin{aligned}
& \frac{1}{\Delta t^2}(w, \Delta u)_\Omega + \frac{1}{2\Delta t}((\mathbf{a} \cdot \mathbf{n})w, \Delta u)_{\Gamma_{out}} + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega \\
&= \frac{1}{\Delta t}(\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{1}{\Delta t}((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma_{out}} - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega \\
&+ \frac{1}{\Delta t}(w, s)_\Omega + \frac{1}{2}(\mathbf{a} \cdot \nabla w, s)_\Omega
\end{aligned} \tag{9}$$

neglecting the outflow boundary term, because the wave front doesn't reach it, yields,

$$\begin{aligned}
\frac{1}{\Delta t^2}(w, \Delta u)_\Omega + \frac{1}{4}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla \Delta u)_\Omega &= \frac{1}{\Delta t}(\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{1}{2}(\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u^n)_\Omega \\
&+ \frac{1}{\Delta t}(w, s)_\Omega + \frac{1}{2}(\mathbf{a} \cdot \nabla w, s)_\Omega
\end{aligned} \tag{10}$$

which is written in discrete form as:

$$\frac{1}{\Delta t^2} \mathbf{M} \Delta \mathbf{U} + \frac{1}{4} \mathbf{K} \Delta \mathbf{U} = \frac{1}{\Delta t} \mathbf{C} \mathbf{U}^n - \frac{1}{2} \mathbf{K} \mathbf{U}^n + \frac{1}{\Delta t} \mathbf{v}_1 + \frac{1}{2} \mathbf{v}_2 \tag{11}$$

where the appearing vectors are defined as:

$$\begin{aligned}
\mathbf{M} \mathbf{U} &= (w, u)_\Omega, \\
\mathbf{C} \mathbf{U} &= (\mathbf{a} \cdot \nabla w, u)_\Omega, \\
\mathbf{K} \mathbf{U} &= (\mathbf{a} \cdot \nabla w, \mathbf{a} \cdot \nabla u)_\Omega, \\
\mathbf{v}_1 &= (w, s)_\Omega, \\
\mathbf{v}_2 &= (\mathbf{a} \cdot \nabla w, s)_\Omega,
\end{aligned} \tag{12}$$

equation (11) is further simplified to:

$$\left[\mathbf{M} + \frac{\Delta t^2}{4} \mathbf{K} \right] \Delta \mathbf{U} = \left[\Delta t \mathbf{C} - \frac{\Delta t^2}{2} \mathbf{K} \right] \mathbf{U}^n + \left[\Delta t \mathbf{v}_1 + \frac{\Delta t^2}{2} \mathbf{v}_2 \right] \tag{13}$$

and for ease of implementation in Matlab, it is written as:

$$\mathbf{A} \Delta \mathbf{U} = \mathbf{B} \mathbf{U}^n + \mathbf{f}$$

and the corresponding part of the Matlab function (`system_CN_LS.m`) is:

```

1 % Matrices assembly
2 A(isp,isp) = A(isp,isp) + w_ig*(N'*N + dt_2^2*(a*Nx)'*(a*Nx));
3 B(isp,isp) = B(isp,isp) + w_ig*(dt*(a*Nx)'*N - (dt^2/2)*(a*Nx)'*(a*Nx));
4 f(isp) = f(isp) + w_ig*(dt*(N')*SourceTerm(x) + ...
   (dt^2/2)*(a*Nx)'*SourceTerm(x));

```

Now, using the least-square formulation (linear finite elements) with Crank-Nicholson time integration scheme gives the solution shown in Figure 4b. It is observed that Crank-Nicholson with least-squares succeeded in removing the spurious oscillation induced by the Galerkin formulation over the whole computational domain.

There still exist some residual oscillations at the front because Crank-Nicholson scheme is not a monotone scheme [1]. These could be removed by adding more artificial viscosity that locally (only) at the front.

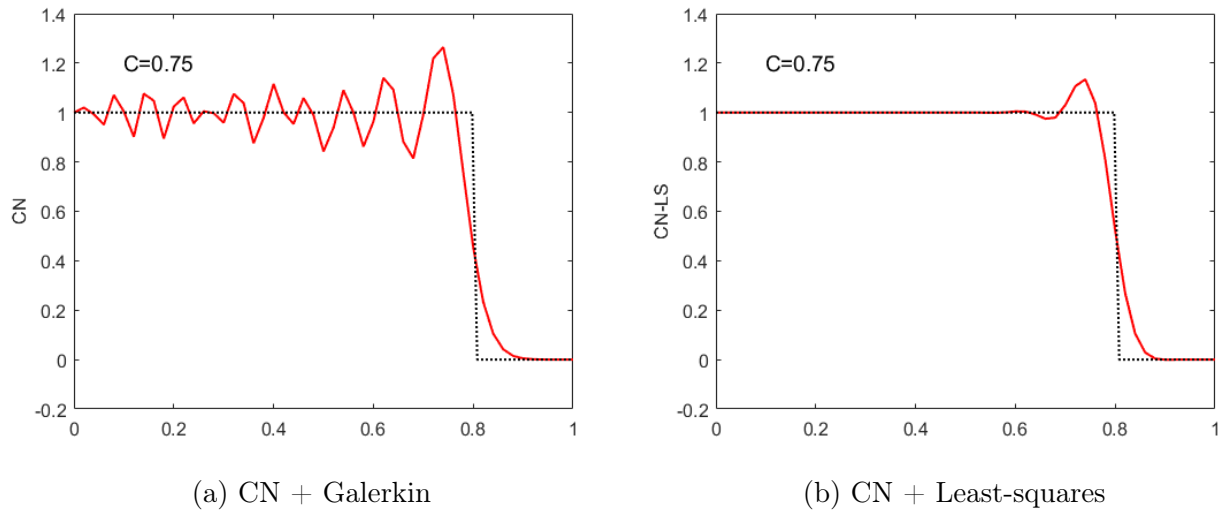


Figure 4: Propagation of steep front using the Crank-Nicholson scheme with Galerkin formulation (left) and with the least-squares formulation (right).

1.2.3 Lax-Wendroff with linear finite elements for Galerkin formulation

Solving the problem using the second-order Lax-Wendroff scheme with consistent mass matrix (TG2) yields unstable solution as seen in Figure 5. The reason is that Lax-Wendroff is conditionally stable, where the stability condition is $C^2 < \frac{1}{3}$, i.e. $|C| < 0.5774$.

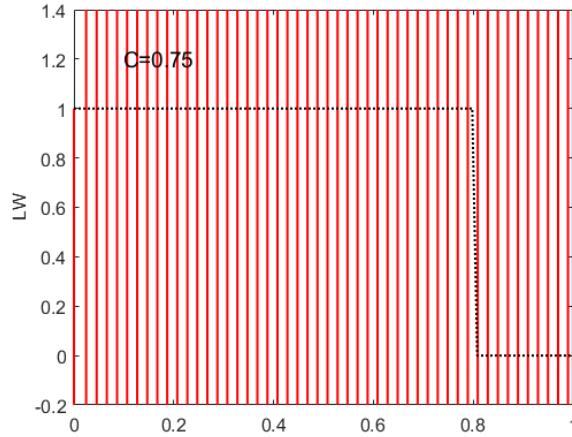


Figure 5: Propagation of step front using the second-order Lax-Wendroff scheme with consistent mass matrix (TG2) and with the Galerkin method.

A way to increase the stability limit to $|C| < 1$ is to use lumped mass matrix. The results using the second-order Lax-Wendroff scheme with lumped mass matrix (LW-FD) yields stable solution as seen in Figure 6.

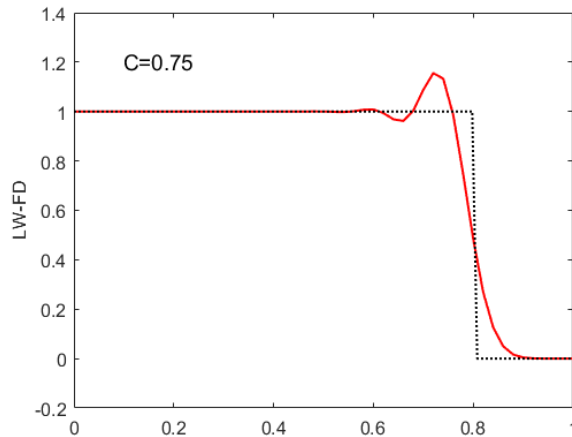


Figure 6: Propagation of step front using the second-order Lax-Wendroff scheme with lumped mass matrix (LW-FD) and with the Galerkin method.

1.2.4 Two-step Lax-Wendroff with linear finite elements for Galerkin formulation

The time discretization of the second-order two-step Lax-Wendroff scheme, also called Richtmyer scheme, is given by:

$$\begin{aligned} u^{n+\frac{1}{2}} &= u^n + \frac{\Delta t}{2} u_t^n, \\ u^{n+1} &= u^n + \Delta t u_t^{n+\frac{1}{2}} \end{aligned} \quad (14)$$

recalling the linear advection equation:

$$u_t^n = s^n - \mathbf{a} \cdot \nabla u^n \quad (15)$$

substituting (15) into (14) yields:

$$u^{n+\frac{1}{2}} = u^n + \frac{\Delta t}{2} s^n - \frac{\Delta t}{2} \mathbf{a} \cdot \nabla u^n \quad (16a)$$

$$u^{n+1} = u^n + \Delta t s^{n+\frac{1}{2}} - \Delta t \mathbf{a} \cdot \nabla u^{n+\frac{1}{2}} \quad (16b)$$

assuming that the source term is not time dependent, then the Galerkin weak form is given by:

$$(w, u^{n+\frac{1}{2}})_\Omega = (w, u^n)_\Omega + \frac{\Delta t}{2} (w, s)_\Omega - \frac{\Delta t}{2} (w, \mathbf{a} \cdot \nabla u^n)_\Omega \quad (17a)$$

$$(w, u^{n+1})_\Omega = (w, u^n)_\Omega + \Delta t (w, s)_\Omega - \Delta t (w, \mathbf{a} \cdot \nabla u^{n+\frac{1}{2}})_\Omega \quad (17b)$$

integration by parts and applying Dirchlet boundary condition at the inlet yields:

$$\begin{aligned} (w, u^{n+\frac{1}{2}})_\Omega &= (w, u^n)_\Omega + \frac{\Delta t}{2} (w, s)_\Omega + \frac{\Delta t}{2} (\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{\Delta t}{2} ((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma_{out}} \\ &\quad - \frac{\Delta t}{2} ((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma_N^{in}} \end{aligned} \quad (18a)$$

$$\begin{aligned} (w, u^{n+1})_\Omega &= (w, u^n)_\Omega + \Delta t (w, s)_\Omega + \Delta t (\mathbf{a} \cdot \nabla w, u^{n+\frac{1}{2}})_\Omega - \Delta t ((\mathbf{a} \cdot \mathbf{n})w, u^{n+\frac{1}{2}})_{\Gamma_{out}} \\ &\quad - \Delta t ((\mathbf{a} \cdot \mathbf{n})w, u^{n+\frac{1}{2}})_{\Gamma_N^{in}} \end{aligned} \quad (18b)$$

particularizing it to the problem at hand (no Neumann boundary) yields:

$$(w, u^{n+\frac{1}{2}})_\Omega = (w, u^n)_\Omega + \frac{\Delta t}{2} (w, s)_\Omega + \frac{\Delta t}{2} (\mathbf{a} \cdot \nabla w, u^n)_\Omega - \frac{\Delta t}{2} ((\mathbf{a} \cdot \mathbf{n})w, u^n)_{\Gamma_{out}} \quad (19a)$$

$$(w, u^{n+1})_\Omega = (w, u^n)_\Omega + \Delta t (w, s)_\Omega + \Delta t (\mathbf{a} \cdot \nabla w, u^{n+\frac{1}{2}})_\Omega - \Delta t ((\mathbf{a} \cdot \mathbf{n})w, u^{n+\frac{1}{2}})_{\Gamma_{out}} \quad (19b)$$

neglecting the outflow boundary term, because the wave front doesn't reach it, yields, particularizing it to the problem at hand (no Neumann boundary) yields:

$$(w, u^{n+\frac{1}{2}})_\Omega = (w, u^n)_\Omega + \frac{\Delta t}{2} (w, s)_\Omega + \frac{\Delta t}{2} (\mathbf{a} \cdot \nabla w, u^n)_\Omega \quad (20a)$$

$$(w, u^{n+1})_\Omega = (w, u^n)_\Omega + \Delta t (w, s)_\Omega + \Delta t (\mathbf{a} \cdot \nabla w, u^{n+\frac{1}{2}})_\Omega \quad (20b)$$

recalling the definitions given by (12), the discrete form is written as:

$$MU^{n+\frac{1}{2}} = MU^n + \frac{\Delta t}{2}v_1 + \frac{\Delta t}{2}CU^n \quad (21a)$$

$$MU^{n+1} = MU^n + \Delta tv_1 + \Delta tCU^{n+\frac{1}{2}} \quad (21b)$$

further simplification yields:

$$M(U^{n+\frac{1}{2}} - U^n) = \frac{\Delta t}{2}CU^n + \frac{\Delta t}{2}v_1 \quad (22a)$$

$$M(U^{n+1} - U^n) = \Delta tCU^{n+\frac{1}{2}} + \Delta tv_1 \quad (22b)$$

and for ease of implementation in Matlab, it is written as:

$$A(U^{n+\frac{1}{2}} - U^n) = \frac{1}{2}BU^n + \frac{1}{2}f, \quad (23a)$$

$$A(U^{n+1} - U^n) = BU^{n+\frac{1}{2}} + f \quad (23b)$$

and the corresponding part of the Matlab function (`system_LW_2S.m`) is:

```

1 % Matrices assembly
2 A(isp,isp) = A(isp,isp) + w_ig*N'*N;
3 B(isp,isp) = B(isp,isp) + w_ig*dt*(a*Nx)'*N;
4 f(isp) = f(isp) + w_ig*dt_2*(N')*SourceTerm(x);

```

and the modification made in the `Main.m` to account for the two-steps is as follows:

```

1 % SOLUTION AT EACH TIME STEP
2 if meth == 7 || meth == 8 % two-step Lax-Wendroff methods
3     for n = 1:nstep
4         % step 1
5         btot_1 = [0.5*B*u(:,n)+0.5*f; bccd];
6         aux_1 = U\ (L\btot_1);
7         u(:,n+1) = u(:,n) + aux_1(1:numnp);
8         % step 2
9         btot_2 = [B*u(:,n+1)+f; bccd];
10        aux_2 = U\ (L\btot_2);
11        u(:,n+1) = u(:,n) + aux_2(1:numnp);
12    end
13 else

```

Using the two-step Lax-Wendroff method with consistent mass matrix yields unstable solution as seen in Figure 7a. This is expected because the stability limit is exceeded, where the Courant number is $C = 0.75$ while the limit is $|C| < 0.5774$. The solution for the same method but with lumped mass matrix is shown in Figure 7b. It is observed that the stability is better, but the solution is still not satisfactory. In fact, the stability limit of the two-step Lax-Wendroff method applied to the steep-front problem was found to be extremely reduced, see Figure 8 where $C = 0.03, 0.0003$ and yet the solution is not improving.

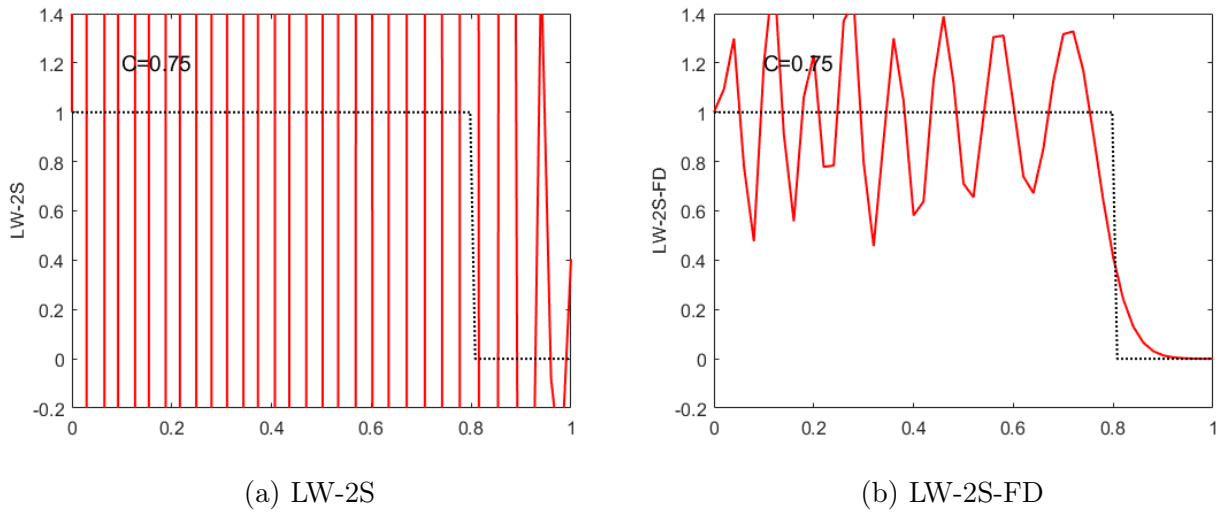


Figure 7: Propagation of step front using the two-step Lax-Wendroff time integration scheme with consistent mass matrix (left) and with lumped mass matrix (right) combined with Galerkin spatial formulation.

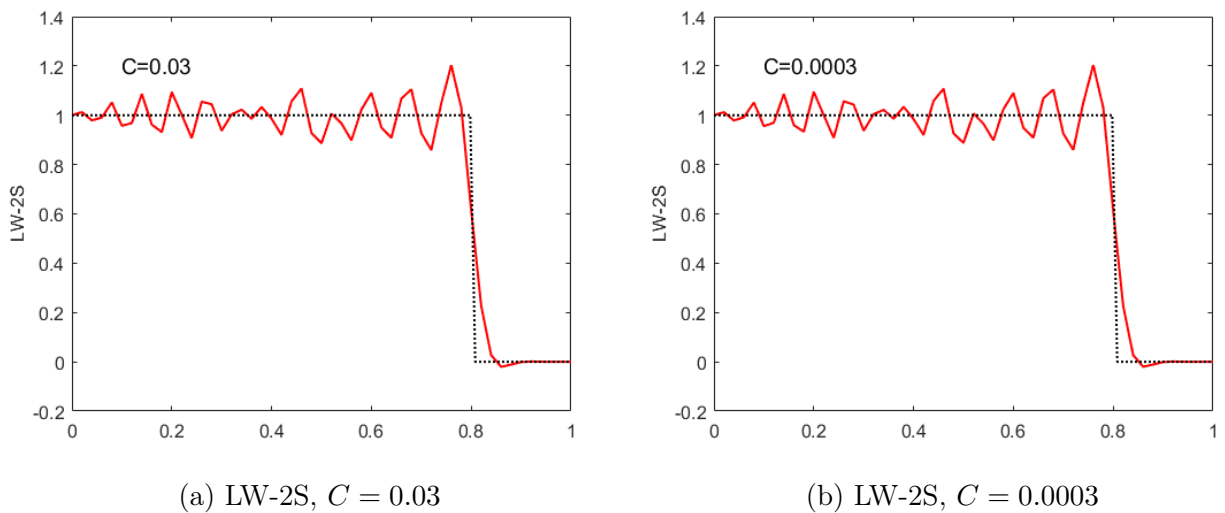


Figure 8: Propagation of step front using the two-step Lax-Wendroff time integration scheme with consistent mass matrix combined with Galerkin spatial formulation at very small values of Courant number.

2 1D unsteady convection-diffusion

$$\begin{cases} u_t + au_x - \nu u_{xx} = 0 & x \in (0, 1), t \in (0, 0.6] \\ u(x, 0) = u_0(x) & x \in (0, 1) \\ u(0, t) = u(1, t) = 0 & t \in (0, 0.6] \end{cases}$$

$$u_0(x) = \frac{5}{7} \exp\{-(x - x_0)^2/L^2\}$$

$$a = 1, x_0 = 2/15, L = 7\sqrt{2}/300, \Delta x = 1/150$$

$$\nu = [3.3 \cdot 10^{-3}, 6.7 \cdot 10^{-4}, 3.3 \cdot 10^{-5}]^T$$

2.1 Galerkin and Crank-Nicholson

The solution is shown in Figure 9 at Courant number $C = 1$. It is noticed that the second-order Crank-Nicholson scheme gives good results at low and moderate values of Peclet number as in Figures 9a and 9b, but shows significant phase error when the Peclet number is increased, i.e. convection dominated cases as in Figure 9c, and it becomes more worse as the Courant number exceeds 1 as in Figure 9d. The reason is that linear finite elements in standard Galerkin formulation do not work well with the second-order Crank-Nicholson scheme in convection dominated problems.

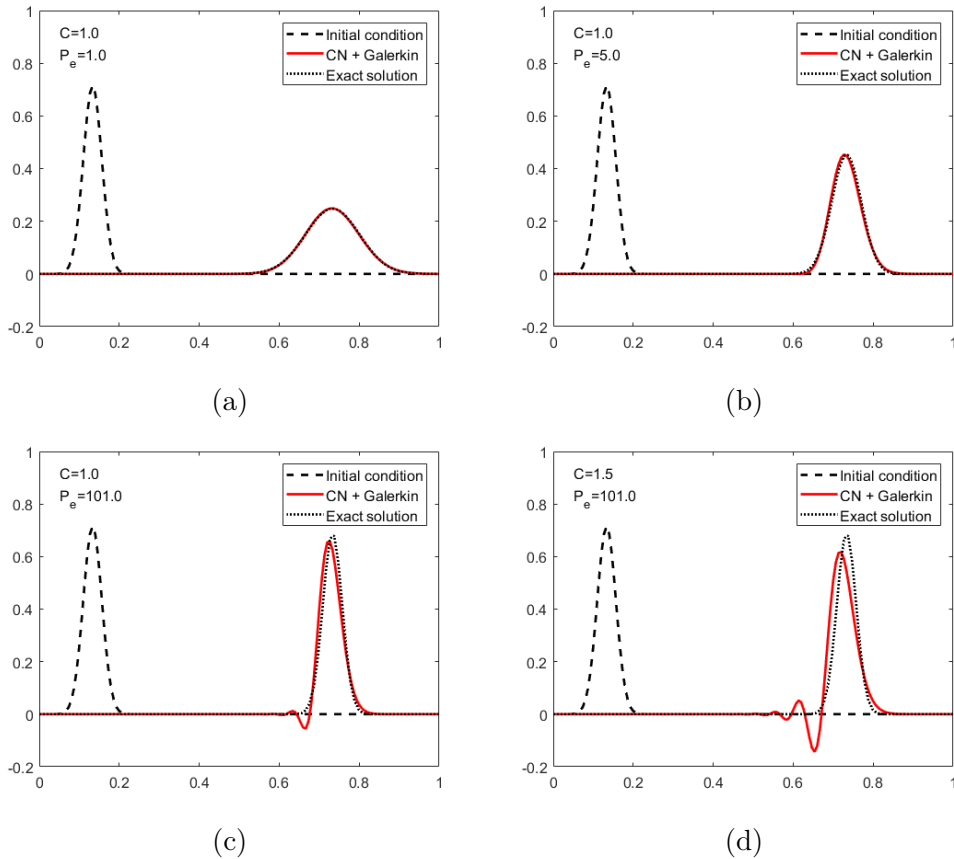


Figure 9: Gaussian hill: standard Galerkin combined with second-order Crank-Nicholson.

2.2 Galerkin and $R_{2,2}$

The solution is shown in Figure 10 at higher Courant number $C = 3$. It is noticed that the higher-order schemes in time provide a gain in accuracy even for convection dominated cases (high Peclet number). It is also noted that the phase accuracy is reduced as the Courant number increases as seen in Figure 10d.

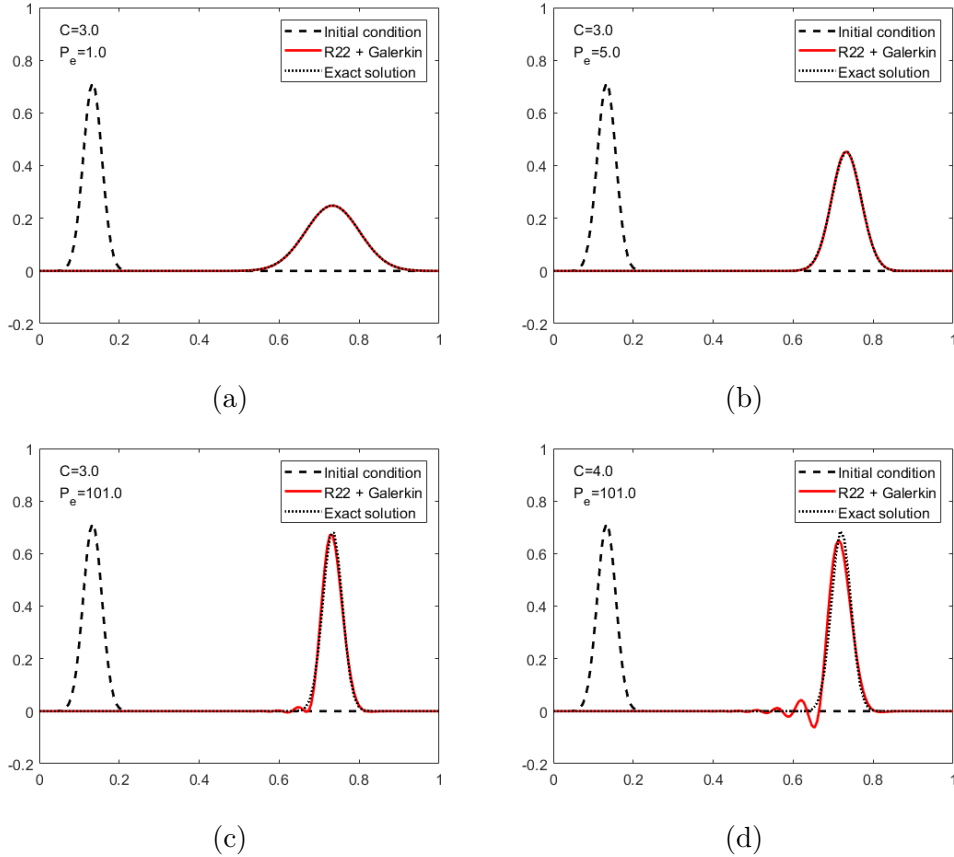


Figure 10: Gaussian hill: standard Galerkin combined with fourth-order $R_{2,2}$.

2.3 Galerkin and $R_{3,3}$

The solution is shown in Figure 11 at even higher Courant number $C = 4$. Again, It is noticed that the higher-order schemes in time provide a gain in accuracy even for convection dominated cases (high Peclet number). It is also noted that the phase accuracy is not significantly affected as the Courant number increases as seen in Figure 11d.

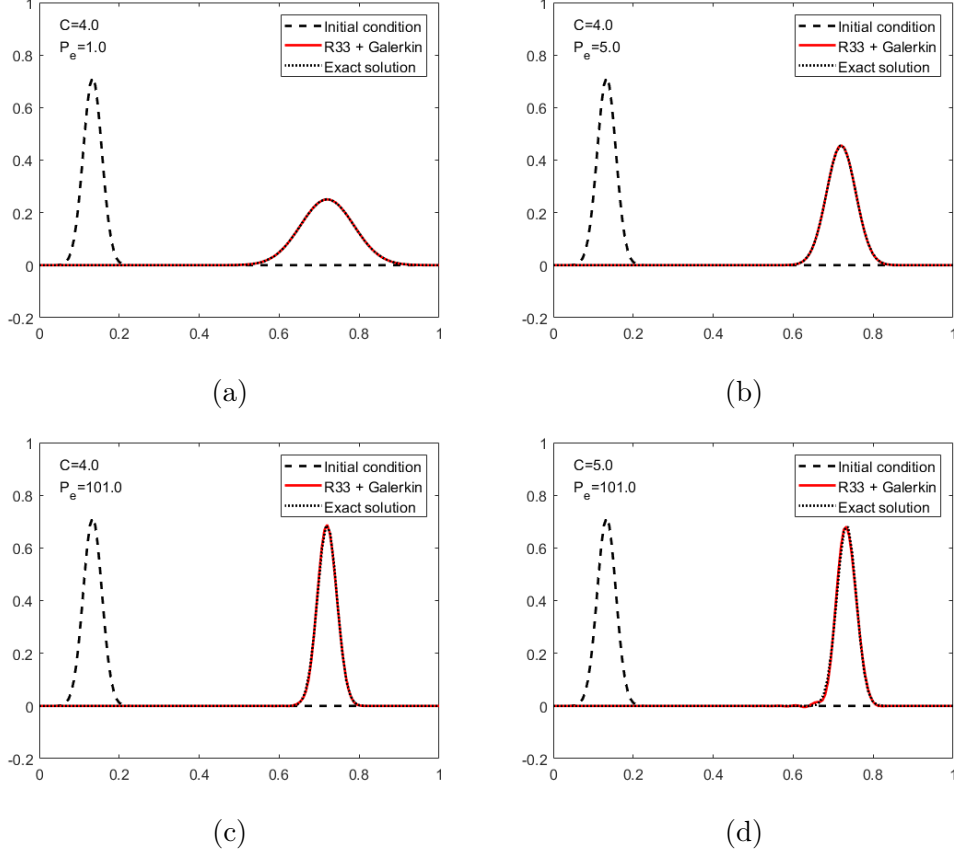


Figure 11: Gaussian hill: standard Galerkin combined with $R_{3,3}$.

2.4 Galerkin and Adams-Bashforth

Adams-Bashforth is a second-order accurate explicit time integration method which is given by:

$$u^{n+1} = u^n + \frac{\Delta t}{2}(3u_t^n - u_t^{n-1}) \quad (24)$$

where $u_t = s - au_x + \nu u_{xx}$. By substituting u_t into the previous equation, the scheme is written for the unsteady convection-diffusion problem as:

$$\frac{2}{\Delta t}\Delta u = 3s^n - 3au_x^n + 3\nu u_{xx}^n - s^{n-1} + au_x^{n-1} - \nu u_{xx}^{n-1} \quad (25)$$

Assuming that the source term is not time dependent yields:

$$\frac{2}{\Delta t}\Delta u = 2s - 3au_x^n + 3\nu u_{xx}^n + au_x^{n-1} - \nu u_{xx}^{n-1} \quad (26)$$

The Galerkin weak form is then written as:

$$\frac{2}{\Delta t}(w, \Delta u)_\Omega = 2(w, s)_\Omega - 3(w, au_x^n)_\Omega + 3(w, \nu u_{xx}^n)_\Omega + (w, au_x^{n-1})_\Omega - (w, \nu u_{xx}^{n-1})_\Omega \quad (27)$$

Integrating by parts the diffusion terms and removing the boundary terms because only Dirichlet boundary conditions exists, the weak form reads:

$$\frac{2}{\Delta t}(w, \Delta u)_\Omega = 2(w, s)_\Omega - 3(w, au_x^n)_\Omega - 3(w_x, \nu u_x^n)_\Omega + (w, au_x^{n-1})_\Omega + (w_x, \nu u_x^{n-1})_\Omega \quad (28)$$

The discrete form is written as:

$$\mathbf{M}\Delta\mathbf{U} = \Delta t(\mathbf{M}\mathbf{s} - \frac{3a}{2}\mathbf{C}\mathbf{U}^n - \frac{3\nu}{2}\mathbf{K}\mathbf{U}^n + \frac{a}{2}\mathbf{C}\mathbf{U}^{n-1} + \frac{\nu}{2}\mathbf{K}\mathbf{U}^{n-1}) \quad (29)$$

where $M_{ij} = (N_i, N_j)_\Omega$, $C_{ij} = (N_i, N_{xj})_\Omega$, and $K_{ij} = (N_{xi}, N_{xj})_\Omega$. Here, the source term is also approximated following the provided code.

It is seen from (29) that Adams-Bashforth is not a self-starting method. For this, a self-starting method is needed in the first time step, the explicit forward Euler method is used with very small time step value. The discrete form of forward Euler scheme for convection-diffusion equation can be shown as:

$$\mathbf{M}\Delta\mathbf{U} = \Delta t(\mathbf{M}\mathbf{s} - a\mathbf{C}\mathbf{U}^n - \nu\mathbf{K}\mathbf{U}^n) \quad (30)$$

and the modification made in the `Main.m` to account for the two-steps is as follows:

```

1  if d_temp == 3 % Adams-Bashforth
2      M_reduced = M(2:end-1, 2:end-1);
3      Sol = c;
4      for i=1:nstep
5          if i == 1 % use forward Euler for 1st step
6              rhs = dt*(M*f - a*C*c - nu*K*c);
7              dc = M_reduced\rhs(2:end-1);
8              c(2:end-1) = c(2:end-1) + dc;
9              Sol = [Sol c];
10         else % use Adams-Bashforth for the rest of steps
11             rhs = dt*(M*f - 1.5*a*C*Sol(:,end) - 1.5*nu*K*Sol(:,end) + ...
12                 0.5*a*C*Sol(:,end-1) + 0.5*nu*K*Sol(:,end-1));
13             dc = M_reduced\rhs(2:end-1);
14             c(2:end-1) = c(2:end-1) + dc;
15             Sol = [Sol c];
16         end
17     else

```

By implementing this method and analysing it, it is noticed that based on the value of Peclet number, this method gives accurate results only at low values of the Courant number, and surprisingly tends to be more unstable for diffusion-dominated case (low values of Pe).

Diffusion-dominated ($Pe = 0.33$): the stability is only achieved at very low values of the Courant number around $C = 0.037$ as in Figure 12a. Slightly increasing the value of the Courant number to $C = 0.06$ yields unstable results in Figure 12b.

Balanced convection-diffusion ($Pe = 1$): the stability is only achieved at low values of the Courant number around $C = 0.1$ as in Figure 12c. Slightly increasing the value of the Courant number to $C = 0.2$ yields unstable results in Figure 12d.

Convection-dominated ($Pe = 5$): Stable solution was obtained at $C = 0.4$ as in Figure 12e. Slightly increasing the Courant number to $C = 0.5$ leads to instabilities in the solution as shown in Figure 12f.

Highly convection-dominated ($Pe = 100$): Stable solution was obtained at $C = 0.3$ as in Figure 12g. Slightly increasing the Courant number to $C = 0.4$ leads to unstable solution as shown in Figure 12h.

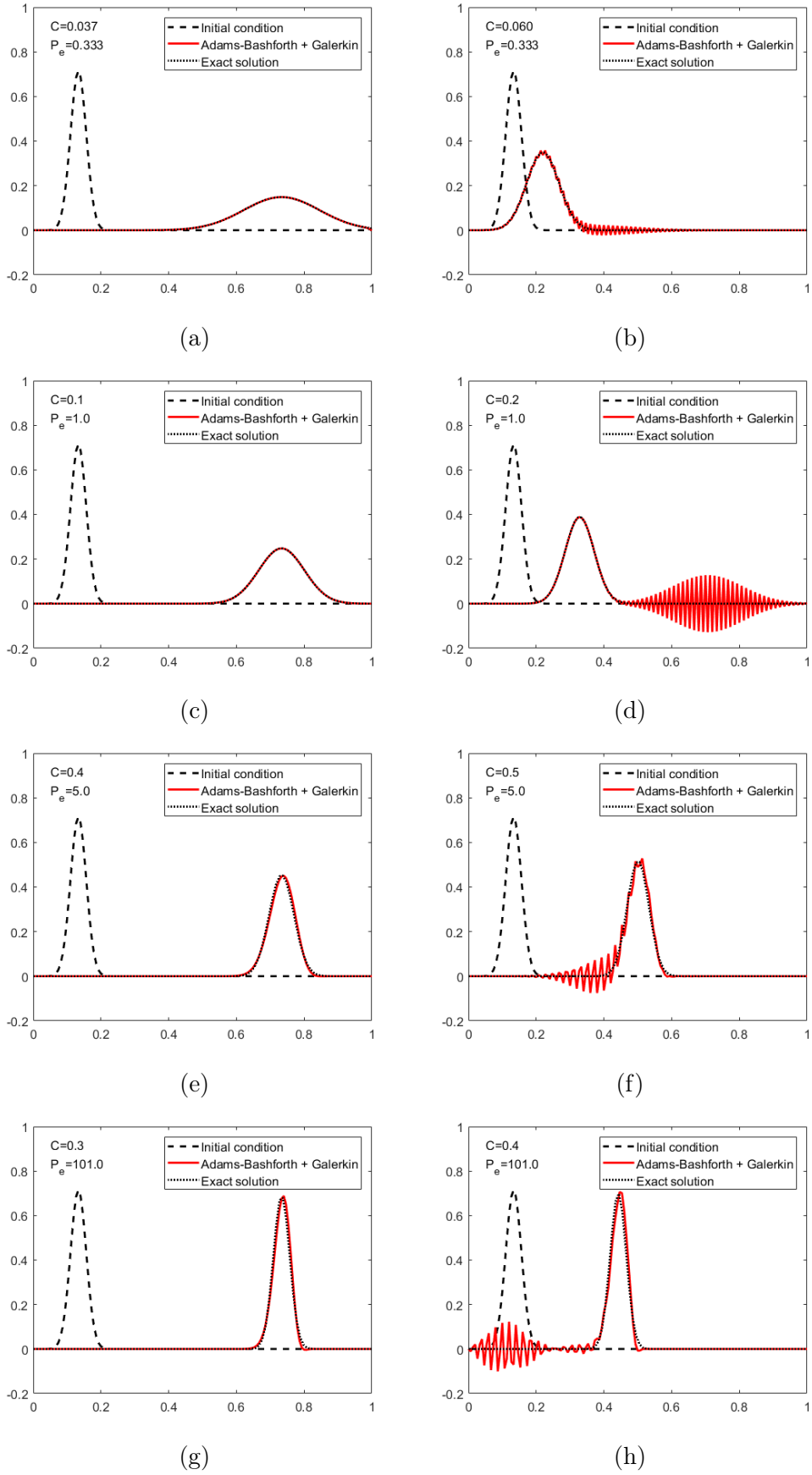


Figure 12: Gaussian hill: standard Galerkin combined with Adams-Bashforth.

2.5 Time-discontinuous Galerkin formulation for the convection-diffusion equation

Following the steps in [2] done to formulate the time-discontinuous Galerkin for the convection equation, similarly, the time-discontinuous Galerkin for the convection-diffusion equation is formulated.

Considering a piecewise continuous approximation in space and discontinuous approximation in time, the time domain is partitioned in n_{st} sub-intervals, where each sub-interval is defined as $I^n =]t^n, t^{n+1}[$, for $n = 0, 1, \dots, n_{st} - 1$.

Space-time slabs are then obtained in the form:

$$Q^n = \Omega \times I^n$$

For the considered space-time slab Q^n , the spatial domain Ω is subdivided into n_{el} elements, Ω^e , $e = 1, \dots, n_{el}$, giving space-time element domains

$$Q_e^n = \Omega^e \times I^n, \quad e = 1, \dots, n_{el}$$

Considering the following notation

$$u^h(t_{\pm}^n) = \lim_{\epsilon \rightarrow 0^+} u^h(t^n \pm \epsilon)$$

The weak form of the convection-diffusion equation with zero source term is written as:

$$\int \int_{Q^n} w^h (u_t^h + \mathbf{a} \cdot \nabla u^h - \nabla \cdot (\nu \nabla u^h)) d\Omega dt + \int_{\Omega} w^h(t_+^n) (u^h(t_+^n) - u^h(t_-^n)) d\Omega = 0 \quad (31)$$

with the initial condition $u^h(\mathbf{x}, t_-^0) = u_0(\mathbf{x})$.

Integrating by parts and considering only Dirichlet boundary conditions yields:

$$\int \int_{Q^n} w^h (u_t^h + \mathbf{a} \cdot \nabla u^h) d\Omega dt + \int \int_{Q^n} \nabla w^h \cdot \nu \nabla u^h d\Omega dt + \int_{\Omega} w^h(t_+^n) (u^h(t_+^n) - u^h(t_-^n)) d\Omega = 0 \quad (32)$$

Using finite element approximations over a space-time slab which are piecewise polynomials in space and linear in time; that is, for $(\mathbf{x}, t) \in Q^n = \Omega \times I^n$,

$$u^h(\mathbf{x}, t) = \sum_{A=1}^{n_{np}} N_A(\mathbf{x}) (\Theta_1(t) \tilde{u}_A^n + \Theta_2(t) u_A^{n+1})$$

where $N_A(\mathbf{x})$ is the spatial shape function at node A ; \tilde{u}_A^n and u_A^{n+1} are the nodal values of u^h for node A at t_+^n and t_-^{n+1} , respectively. $\Theta_1(t)$ and $\Theta_2(t)$ are the time interpolation functions defined linearly as

$$\Theta_1(t) = \frac{t^{n+1} - t}{t^{n+1} - t^n} = \frac{t^{n+1} - t}{\Delta t}$$

$$\Theta_2(t) = \frac{t - t^n}{t^{n+1} - t^n} = \frac{t - t^n}{\Delta t}$$

The test function w^h for each time slab (piecewise polynomials in space and linear in time) are similarly defined, $N_A(\mathbf{x})\Theta_1(t)$ and $N_A(\mathbf{x})\Theta_2(t)$ for $A = 1, \dots, n_{np}$.

With these definitions, the weak form (31) yields the following couple of equations for each node A :

$$\sum_{B=1}^{n_{np}} \left\{ \int \int_{Q^n} N_A \Theta_1 \left[N_B \frac{u_B^{n+1} - \tilde{u}_B^n}{\Delta t} + (\Theta_1 \tilde{u}_B^n + \Theta_2 u_B^{n+1})(\mathbf{a} \cdot \nabla) N_B \right] d\Omega dt \right. \\ \left. + \int \int_{Q^n} \nabla N_A \Theta_1 \nu (\Theta_1 \tilde{u}_B^n + \Theta_2 u_B^{n+1}) \nabla N_B \right\} = 0 \quad (33a)$$

$$\sum_{B=1}^{n_{np}} \left\{ \int \int_{Q^n} N_A \Theta_2 \left[N_B \frac{u_B^{n+1} - \tilde{u}_B^n}{\Delta t} + (\Theta_1 \tilde{u}_B^n + \Theta_2 u_B^{n+1})(\mathbf{a} \cdot \nabla) N_B \right] d\Omega dt \right. \\ \left. + \int \int_{Q^n} \nabla N_A \Theta_2 \nu (\Theta_1 \tilde{u}_B^n + \Theta_2 u_B^{n+1}) \nabla N_B \right\} \quad (33b) \\ + \int_{Q^n} N_A \sum_{B=1}^{n_{np}} N_B (\tilde{u}_B^n - u_B^n) d\Omega = 0$$

Following the Matrix form for the convection equation shown in [3], The Matrix form for the convection-diffusion equation is written, where the unknowns are \mathbf{u}^{n+1} and \mathbf{u}^{n+} , as:

$$\left(\mathbf{M} + \frac{2}{3} \Delta t \mathbf{C} + \frac{2\nu}{3} \Delta t \mathbf{K} \right) \mathbf{u}^{n+1} - \left(\mathbf{M} - \frac{1}{3} \Delta t \mathbf{C} - \frac{\nu}{3} \Delta t \mathbf{K} \right) \mathbf{u}^{n+} = \mathbf{0} \quad (34a)$$

$$\left(\mathbf{M} + \frac{1}{3} \Delta t \mathbf{C} + \frac{\nu}{3} \Delta t \mathbf{K} \right) \mathbf{u}^{n+1} + \left(\mathbf{M} + \frac{2}{3} \Delta t \mathbf{C} + \frac{2\nu}{3} \Delta t \mathbf{K} \right) \mathbf{u}^{n+} = 2\mathbf{M} \mathbf{u}^{n-} \quad (34b)$$

A Developed codes for 1D unsteady convection

A.1 New function system_CN_LS.m

This function includes the implementation of the least-square spatial formulation combined with Crank-Nicolson time integration scheme.

```
1 function [A,B,f] = system_CN_LS(xnode,a)
2 % [A,B,f] = system_CN_LS(xnode,a)
3 % L.h.s (A) and r.h.s (B,f) matrices for the second-order
4 % implicit Crank-Nicolson scheme using the consistent mass matrix.
5 %
6 % Least-Squares spatial discretization is used
7 %
8 % xnode: nodal coordinates
9 % a : velocity
10 %
11
12
13 global dt
14
15 dt_2 = dt/2;
16
17 % Gauss points and weights on the reference element [-1,1]
18 xipg = [-1/sqrt(3) 1/sqrt(3)]';
19 wpg = [1 1]';
20
21 % Shape functions and its derivatives in the reference element
22 N_mef = [(1-xipg)/2 (1+xipg)/2];
23 Nxi_mef = [-1/2 1/2; -1/2 1/2];
24
25 % Total number of nodes and elements
26 numnp = size(xnode,2);
27 numel = numnp-1;
28
29 % Number of Gauss points on an element
30 ngaus = size(wpg,1);
31
32 % Allocate storage
33 A = zeros(numnp,numnp);
34 B = zeros(numnp,numnp);
35 f = zeros(numnp,1);
36
37 % MATRICES COMPUTATION
38 % Loop on elements
39 for i=1: numel
40     unos = ones(ngaus,1);
41     h = xnode(i+1)-xnode(i);
42     xm = (xnode(i)+xnode(i+1))/2;
43     weight = wpg*h/2;
44     isp = [i i+1];
45     % Loop on Gauss points (numerical quadrature)
46     for ig=1:ngaus
47         N = N_mef(ig,:);
48         Nx = Nxi_mef(ig,:)*2/h;
49         w_ig = weight(ig);
```

```

50     x = xm + h/2*xipg(ig); % x-coordinate of the current Gauss point
51     % Matrices assembly
52     A(isp,isp) = A(isp,isp) + w_ig*(N'*N + dt_2^2*(a*Nx)'*(a*Nx));
53     B(isp,isp) = B(isp,isp) + w_ig*(dt*(a*Nx)'*N - ...
54         (dt^2/2)*(a*Nx)'*(a*Nx));
54     f(isp) = f(isp) + w_ig*(dt*(N')*SourceTerm(x) + ...
55         (dt^2/2)*(a*Nx)'*SourceTerm(x));
55     end
56 end

```

A.2 New function system_LW_2S.m

This function includes the implementation of the two-step Lax-Wendroff time integration scheme with Galerkin spatial formulation.

```

1  function [A,B,f] = system_LW_2S(xnode,a)
2  % [A,B,f] = system_LW_2S(xnode,a)
3  % L.h.s (A) and r.h.s (B,f) matrices for the two-step Lax-Wendroff scheme
4  % with consistent mass matrix
5  %
6  % xnode: nodal coordinates
7  % a :    velocity
8  %
9
10
11  global dt
12
13  dt_2 = dt/2;
14
15  % Gauss points and weights on the reference element [-1,1]
16  xipg = [-1/sqrt(3) 1/sqrt(3)]';
17  wpg = [1 1]';
18
19  % Shape functions and its derivatives in the reference element
20  N_mef = [(1-xipg)/2 (1+xipg)/2];
21  Nxi_mef = [-1/2 1/2; -1/2 1/2];
22
23  % Total number of nodes and elements
24  numnp = size(xnode,2);
25  numel = numnp-1;
26
27  % Number of Gauss points on an element
28  ngaus = size(wpg,1);
29
30  % Allocate storage
31  A = zeros(numnp,numnp);
32  B = zeros(numnp,numnp);
33  f = zeros(numnp,1);
34
35  % MATRICES COMPUTATION
36  % Loop on elements
37  for i=1: numel
38     unos = ones(ngaus,1);
39     h = xnode(i+1)-xnode(i);
40     xm = (xnode(i)+xnode(i+1))/2;

```

```

41 weight = wpg*h/2;
42 isp = [i i+1];
43 % Loop on Gauss points (numerical quadrature)
44 for ig = 1:ngaus
45     N = N_mef(ig, :);
46     Nx = Nxi_mef(ig, :)*2/h;
47     w_ig = weight(ig);
48     x = xm + h/2*xipg(ig); % x-coordinate of the current Gauss point
49     % Matrices assembly
50     A(isp,isp) = A(isp,isp) + w_ig*N'*N;
51     B(isp,isp) = B(isp,isp) + w_ig*dt*(a*Nx)'*N;
52     f(isp) = f(isp) + w_ig*dt_2*(N')*SourceTerm(x);
53 end
54 end

```

B Developed codes for 1D unsteady convection-diffusion

B.1 Modified function Galerkin.m

This function includes the implementation of Adams-Bashforth time integration scheme with Galerkin spatial formulation.

```
1 function Sol = Galerkin(T,s,a,nu,f,K,M,C,xnode,dt,nstep,c,Accd1,bccd1,d_temp)
2 % Sol = Galerkin(T,s,a,nu,f,K,M,G,xnode,dt,nstep,c,Accd1,bccd1)
3 % This function computes solution Sol at each time step using Galerkin ...
   formulation
4 %
5 % Input
6 %   T,s:           time-integration matrices
7 %   tau:           stabilization matrix
8 %   a,nu:          problem coefficients
9 %   f,K,M,C:       matrices obtained by discretizing the different terms ...
   of the PDE using FEM
10 %   xnode:        vector of nodal coordinates
11 %   dt:           time step
12 %   nstep:        number of time steps to be computed
13 %   c:            initial condition
14 %   Accd1, bccd1: matrices to impose boundary conditions using lagrange ...
   multipliers
15 %   d_temp:       Time discretization method (added for Adams-Bashforth ...
   scheme)
16
17 % Number of points
18 npoin = size(xnode,2);
19
20 if d_temp == 3 % Adams-Bashforth
21     M_reduced = M(2:end-1,2:end-1);
22     Sol = c;
23     for i=1:nstep
24         if i == 1 % use forward Euler for 1st step
25             rhs = dt*(M*f - a*C*c - nu*K*c);
26             dc = M_reduced\rhs(2:end-1);
27             c(2:end-1) = c(2:end-1) + dc;
28             Sol = [Sol c];
29         else % use Adams-Bashforth for the rest of steps
30             rhs = dt*(M*f - 1.5*a*C*Sol(:,end) - 1.5*nu*K*Sol(:,end) + ...
31                 0.5*a*C*Sol(:,end-1) + 0.5*nu*K*Sol(:,end-1));
32             dc = M_reduced\rhs(2:end-1);
33             c(2:end-1) = c(2:end-1) + dc;
34             Sol = [Sol c];
35         end
36     end
37 else
38     % Integration matrix
39     [n,m] = size(T);
40     Id = eye(n,m);
41
42     % Computation of the matrix necessary to obtain solution at each ...
43     time-step: A du = F
44     Kt = a*C + nu*K;
```

```

44 A = [];
45 for i = 1:n
46     row = [];
47     for j = 1:m
48         row = [row, Id(i,j)*M + dt*T(i,j)*Kt];
49     end
50     A = [A; row];
51 end
52
53 Mf = M*f;
54
55 nccd = size(Accd1,1);
56 Accd = []; bccd = [];
57 for i = 1:n
58     row = [];
59     for j = 1:m
60         row = [row, Id(i,j)*Accd1];
61     end
62     Accd = [Accd; row];
63     bccd = [bccd; bccd1];
64 end
65
66 nccd = n*nccd;
67 Atot = [A Accd'; Accd zeros(nccd)];
68
69 % Factorization of matrix Atot
70 [L,U] = lu(Atot);
71
72 Sol = c;
73 % Loop to compute the transient solution
74 for i=1:nstep
75     aux = dt*(-Kt*c + Mf);
76     F = [];
77     for i =1:n
78         F = [F; s(i)*aux];
79     end
80     F = [F;bccd*0];
81     dc = U \ (L\F);
82     dc = reshape(dc(1:n*npoin), npoin, n);
83     c = c + sum(dc,2);
84     Sol = [Sol c];
85 end
86 end

```


References

- [1] Donea, J., Huerta, A. (2004). *Finite Element Methods for Flow Problems*. Chichester: Wiley, pp.141.
- [2] Donea, J., Huerta, A. (2004). *Finite Element Methods for Flow Problems*. Chichester: Wiley, pp.128-129.
- [3] Donea, J., Huerta, A. (2004). *Finite Element Methods for Flow Problems*. Chichester: Wiley, pp.143.