

# **ASSIGNMENT 5**

## **FINITE ELEMENT FOR FLUIDS**



**PRADEEP KUMAR BAL**

**Propagation of a steep front**

## Problem Statement:-

To solve a one-dimensional transient pure convection problem

$$u_t + au_x = 0; a = 1; t \in (0, 0.6]; x \in (0, 1)$$

$S=0$  for the given problem

With the given initial

$$u_0(x) = \{1 \text{ if } x \leq 0.2 ; 0 \text{ Else}$$

And the homogeneous Dirichlet inflow boundary condition which is given as  $u(0, t) = 1$

### 1. Courant Number:

$$C = \frac{|a|\Delta t}{\Delta x} = 0.75$$

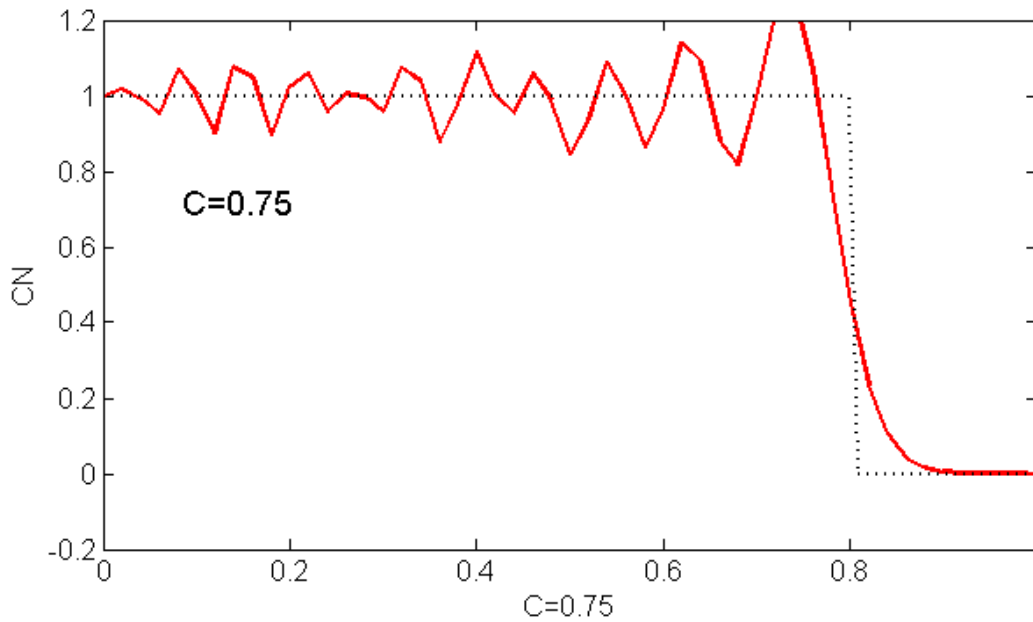
### 2. Crank-Nicholson scheme in time and the Galerkin formulation in space

$$\frac{u^{n+1} - u^n}{\Delta t} + \frac{1}{2} (\mathbf{a} \cdot \nabla)(u^{n+1} - u^n) = -\mathbf{a} * \nabla u^n \text{ (CN Scheme)}$$

#### Weighted Residual and Galerkin Formulation:-

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

The given 1D problem considers the convection at unit speed of discontinuous initial data. The discontinuity occurs over one element and is initially located at position  $x = 0.2$  of the computational domain  $(0, 1)$ . The given inlet condition is imposed. A mesh of uniform linear elements of size  $h=0.02$  is employed. The results at time  $t = 0.6$  are displayed in Figure 1 together with the exact solution. They were obtained (for a Courant number  $C = 0.75$ ) by combining the Crank—Nicolson scheme (with linear elements) and the Galerkin formulation, It can be observed that the Crank—Nicolson Scheme with Galerkin formulation induces spurious oscillations over the whole computational domain. Since Crank—Nicolson is not a monotone scheme, residual oscillations remain at the front.



**Figure 1:** Propagation of a steep front using the Crank—Nicolson scheme with the Galerkin Method at  $C = 0.75$ . The graphs show the computed solutions at time  $t = 0.60$ , together with the exact solution.

### 3. Crank-Nicholson scheme in time and the least-squares formulation in space

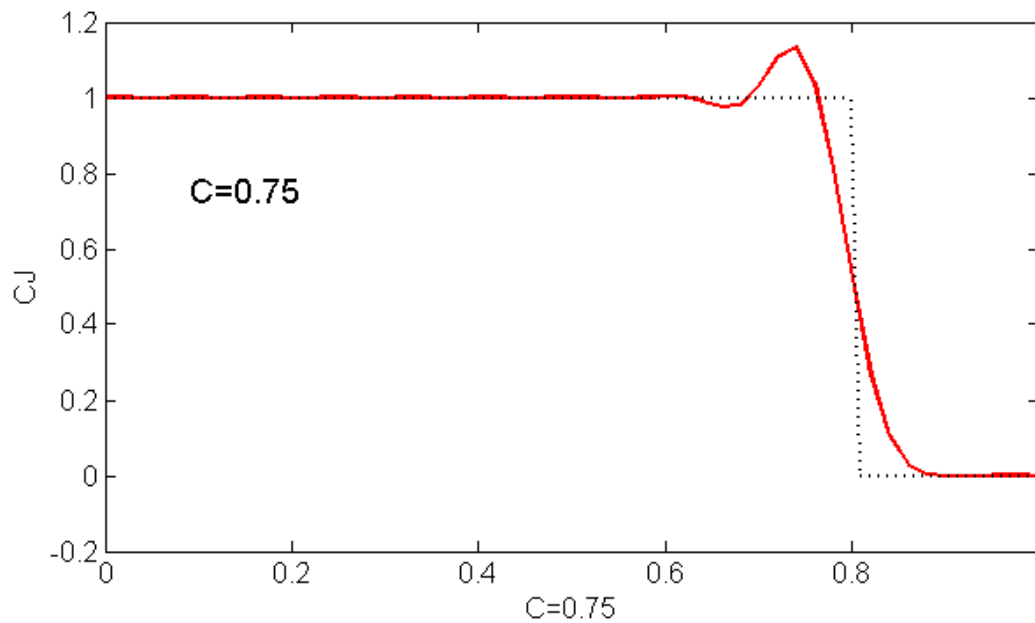
#### Crank-Nicholson scheme in time:

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

This equation can be viewed as a spatial strong form that must be solved at each time step, namely  $L(\Delta u) - f = 0$ . Where  $L = \frac{1}{\Delta t} + \frac{1}{2} (\mathbf{a} \cdot \nabla)$  is the spatial differential operator and  $f = -\mathbf{a} * \nabla u^n$ . Minimization of the least-squares functional,  $(L(\Delta u) - f, L(\Delta u) - f)$ , produces the least square equation  $(L(w), L(\Delta u) - f) = 0$  which takes the following explicit form.

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

The results at time  $t = 0.6$  are displayed in Figure 2 together with the exact solution. They were obtained (for a Courant number  $C = 0.75$ ) by combining the Crank—Nicolson scheme (with linear elements) and the least-squares formulation of Carey and Jiang. It is observed that Crank—Nicolson with least-squares succeeds in removing the spurious oscillations induced by the Galerkin formulation over the whole computational domain.



**Figure 2:** Propagation of a steep front using the Crank—Nicolson scheme with the least-squares method. The Courant number is  $C = 0.75$ . The graphs show the computed solutions at time  $t = 0.60$ , together with the exact solution.

#### 4. Second-order Lax-Wendroff method

##### TG2 Scheme:-

$$\frac{u^{n+1}-u^n}{\Delta t} = u_t^n + \frac{\Delta t}{2} u_{tt}^n + o(\Delta t^2);$$

##### After simplification:-

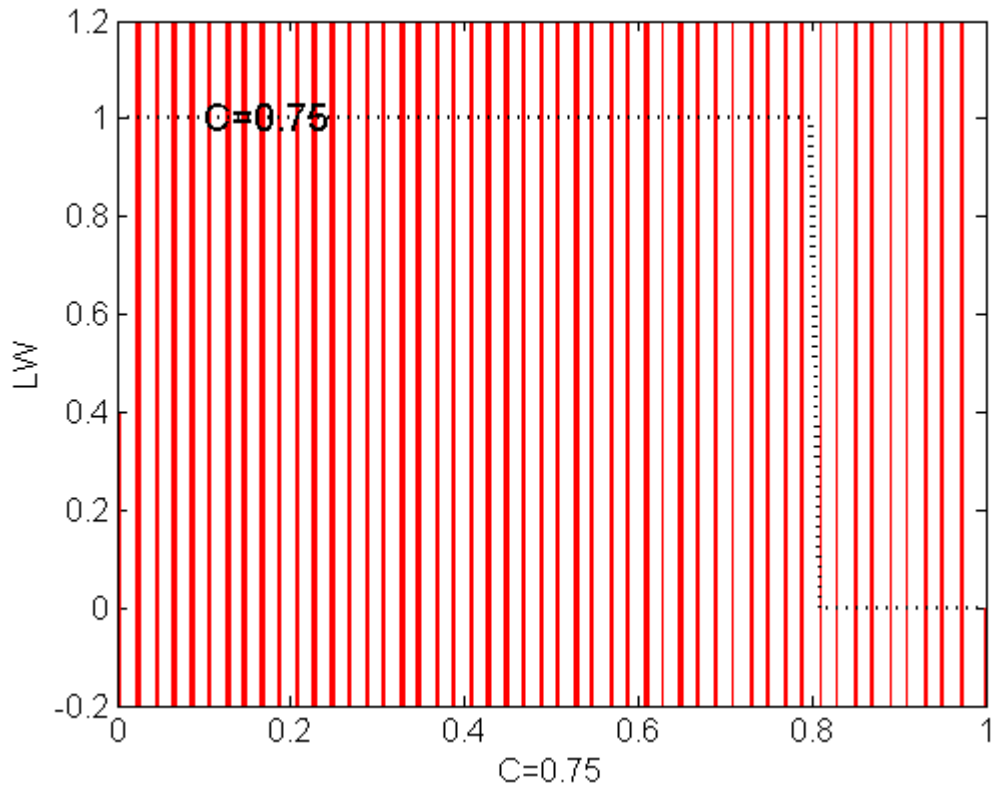
$$\frac{u^{n+1}-u^n}{\Delta t} = -(\mathbf{a} \cdot \nabla) u^n + \frac{\Delta t}{2} (\mathbf{a} \cdot \nabla)^2 u^n$$

##### Galerkin Formulation:-

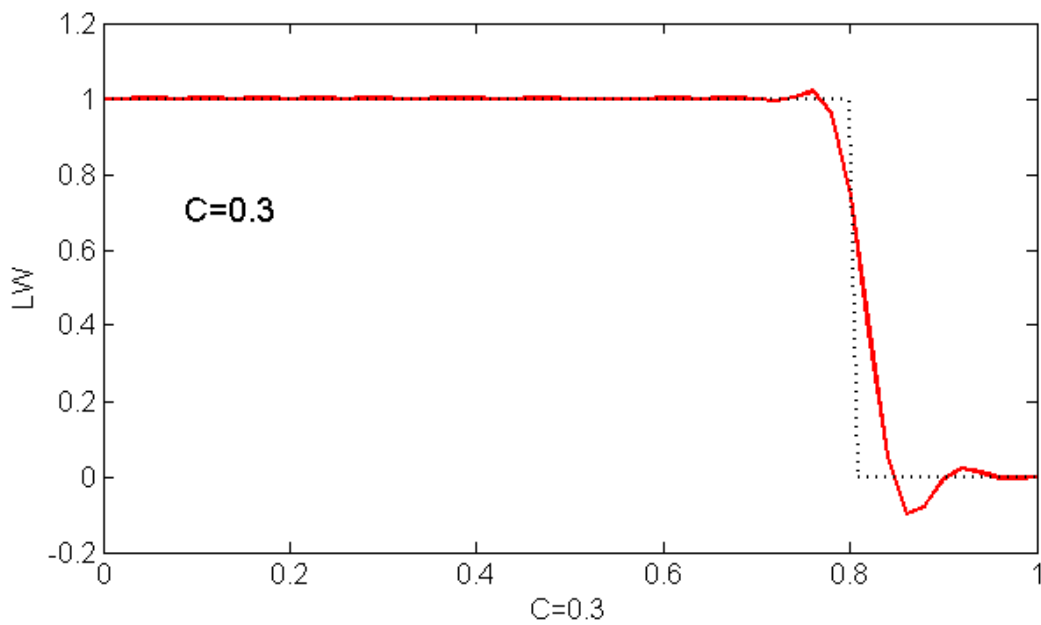
The Galerkin formulation of the given 1D problem for this scheme becomes:

$$\left( w, \frac{\Delta u}{\Delta t} \right) = (aw_x, u^n - \frac{\Delta t}{2} \mathbf{a} \cdot \nabla u^n )$$

The results at time  $t = 0.6$  are displayed in Figure 3 together with the exact solution. They were obtained (for a Courant number  $C = 0.75$ ). As  $C=0.75$ , so the exact solution can't be expected and the method is unstable for  $C=0.75$ . The Lax-Wendroff scheme with consistent mass representation (TG2) cannot be operated with  $C^2 > 1/3$ , Moreover, it shows a phase lead at  $C= 1/2$ . To obtain good results we need to reduce the number of time steps so that the courant number will fall in the stability range of the TG2 scheme. The improved solutions obtained using the TG2 scheme is shown for  $C=0.3$  is shown in the Figure 4.



**Figure 3:** Solutions obtained using the TG2 method for the  $C=0.75$



**Figure 4:** Solutions obtained using the TG2 method for the  $C=0.3$

## 5. TG2-2S

The two-step versions of the explicit Taylor—Galerkin method TG2 which include first time derivatives only and are thus easier to implement than the one-step method TG2.

**Scheme:**

$$u^{n+1/2} = u^n + \frac{\Delta t}{2} u_t^n$$

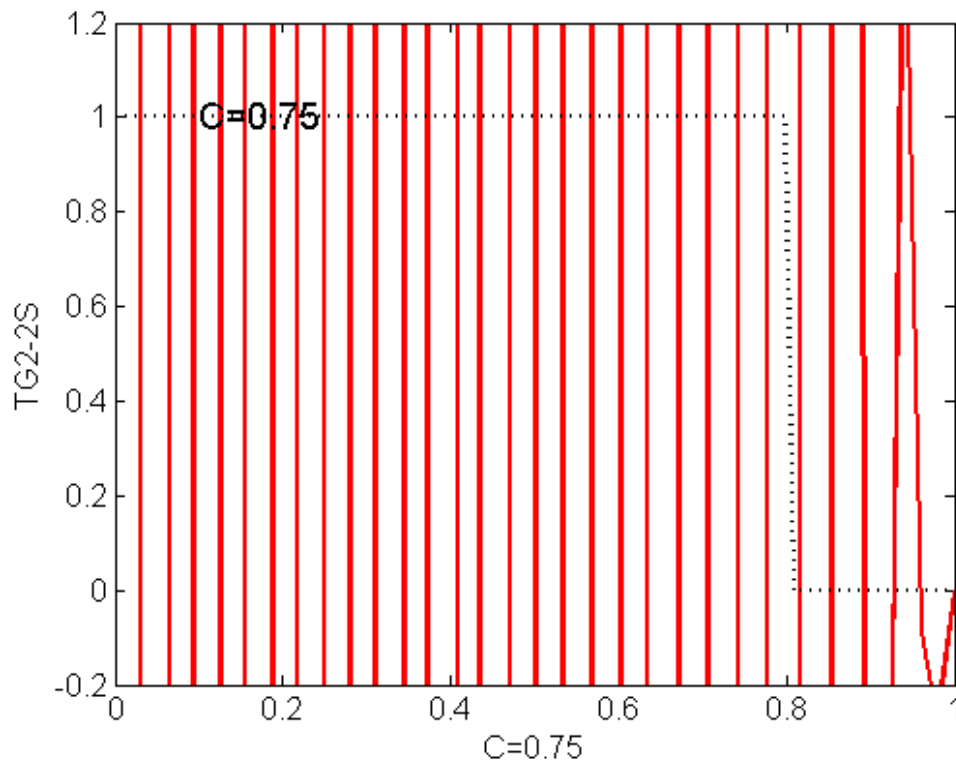
$$u^{n+1} = u^n + \Delta t u^{n+1/2}$$

**Galerkin Formulation for the given problem:-**

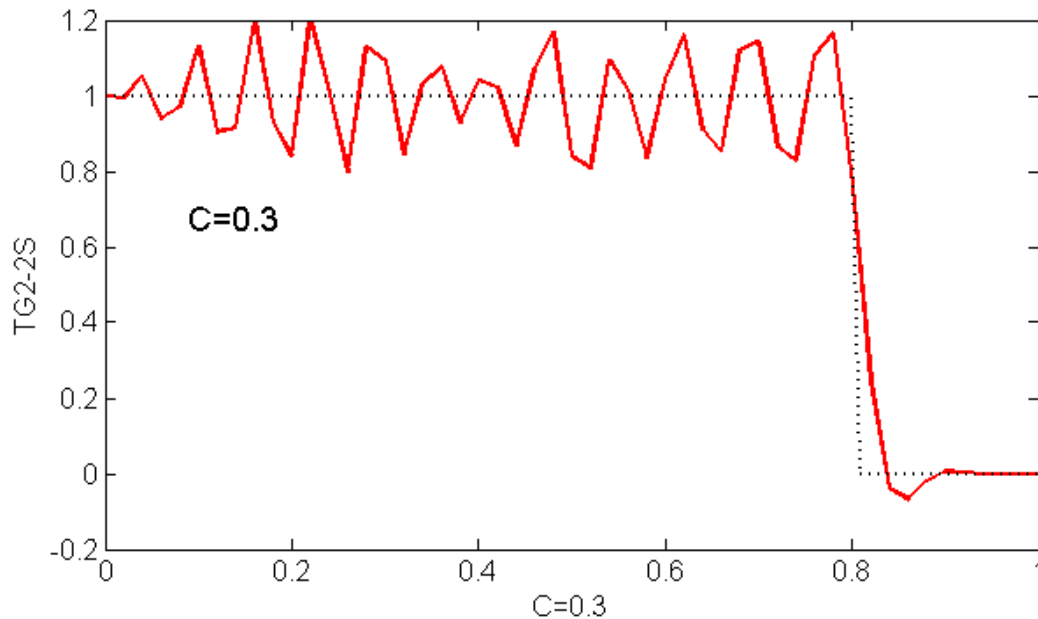
$$\langle w, \frac{u^{n+1/2} - u^n}{\Delta t} \rangle = -\frac{1}{2} \langle w, a u_x^n \rangle$$

$$\langle w, \frac{u^{n+1} - u^n}{\Delta t} \rangle = -\langle w, a u_x^{n+1/2} \rangle$$

This scheme is unstable at  $C=0.75$ . It shows spurious oscillations at  $C=0.3$ . This can be depicted from the Figure 5 and 6. The solutions can be improved using the Discontinuous Galerkin in space and the second-order two-step Lax-Wendroff method in time. The two-step TG2 method integrates in time the semi-discrete equations resulting from the discontinuous Galerkin method.



**Figure 5:** Solutions obtained using the TG2-2S method for the  $C=0.75$



**Figure 6:** Solutions obtained using the TG2-2S ( $C=0.3$ )

## CODE:-

### Implementation of Initial Conditions:

```
% INITIAL CONDITION FOR THE TRANSIENT ANALYSIS
% Steep front
u = zeros(numnp, nstep+1);
x0 = 0.2;
for i=1:numnp
    dist = xnode(i)-x0;
    if dist <= 0
        u(i,1) = 1;
    end
end
end
```

### 1.Crank-Nicholson scheme in time and linear finite element for the Galerkin scheme in space:

```
function [A,B,f] = system_CN(xnode,a)
% [A,B,f] = system_CN(xnode,a)
% L.h.s (A) and r.h.s (B,f) matrices for the second-order
% implicit Crank-Nicolson scheme using the consistent mass matrix.
%
% xnode: nodal coordinates
% a :    velocity
%
global dt
```

```

dt_2 = dt/2;

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';

% Shape functions and its derivatives in the reference element
N_mef = [(1-xipg)/2 (1+xipg)/2];
Nxi_mef = [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points on an element
ngaus = size(wpg,1);

% Allocate storage
A = zeros(numnp,numnp);
B = zeros(numnp,numnp);
f = zeros(numnp,1);

% MATRICES COMPUTATION
% Loop on elements
for i=1:numel
    unos = ones(ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig=1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of the current Gauss point
        % Matrices assembly
        A(isp,isp) = A(isp,isp) + w_ig*(N'*N - dt_2*(a*Nx)'*N);
        B(isp,isp) = B(isp,isp) + w_ig*dt*(a*Nx)'*N;
        f(isp) = f(isp) + w_ig*(N')*SourceTerm(x);
    end
end
end

```

## 2. Crank-Nicholson scheme in time and the least-squares formulation in space (CJ):

```

function [A,B,f] = system_CJ (xnode,a)
% [A,B,f] = system_CJ(xnode,a)
% L.h.s (A) and r.h.s (B,f) matrices for Carey-Jiang method
% Crank-Nicolson method is used for the time-integration whereas
% spatial discretization is performed using linear finite
% elements and the least-squares formulation.
% xnode: nodal coordinates
% a : convection velocity
%

global dt

dt_2 = dt/2;

```



```

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';

% Shape functions on the reference element
N_mef = [(1-xipg)/2 (1+xipg)/2];
Nxi_mef = [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points in each element
ngaus = size(wpg,1);

% Allocate storage
A = zeros(numnp,numnp);
B = zeros(numnp,numnp);
f = zeros(numnp,1);

% MATRICES COMPUTATION
% Loop on elements
for i=1:numel
    unos = ones(ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig=1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
        % Matrices assembly
        A(isp,isp) = A(isp,isp) + w_ig*(N'*N + dt_2*a*(N'*Nx + Nx'*N +
dt_2*a*Nx'*Nx));
        B(isp,isp) = B(isp,isp) - w_ig*dt*a*(N'*Nx + dt_2*a*Nx'*Nx);
        % In this case there is no source term
    end
end
end

```

### 3.TG2(Lax-Wendroff)

```

function [A,B,f] = system_LW(xnode,a)
% [A,B,f] = system_LW(xnode,a)
% L.h.s (A) and r.h.s (B,f) matrices for the second-order
% explicit Taylor-Galerkin method (Lax-Wendroff).
% The spatial discretization is performed using linear finite
% elements and the Galerkin formulation.
% xnode: nodal coordinates
% a : convection velocity
%

global dt

dt_2 = dt/2;

% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';

```

```

wpg = [1 1]';

% Shape functions on the reference element
N_mef = [(1-xipg)/2 (1+xipg)/2];
Nxi_mef = [-1/2 1/2; -1/2 1/2];

% Total number of nodes and elements
numnp = size(xnode,2);
numel = numnp-1;

% Number of Gauss points in each element
ngaus = size(wpg,1);

% Allocate storage
A = zeros(numnp,numnp);
B = zeros(numnp,numnp);
f = zeros(numnp,1);

% MATRICES COMPUTATION
% Loop on elements
for i=1:numel
    unos = ones(ngaus,1);
    h = xnode(i+1)-xnode(i);
    xm = (xnode(i)+xnode(i+1))/2;
    weight = wpg*h/2;
    isp = [i i+1];
    % Loop on Gauss points (numerical quadrature)
    for ig=1:ngaus
        N = N_mef(ig,:);
        Nx = Nxi_mef(ig,:)*2/h;
        w_ig = weight(ig);
        x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
        % Matrices assembly
        A(isp,isp) = A(isp,isp) + w_ig*N'*N;
        B(isp,isp) = B(isp,isp) + w_ig*dt*(a*Nx)'*(N-dt_2*a*Nx);
        f(isp) = f(isp) + dt*w_ig*(dt_2*a*Nx + N)'*SourceTerm(x);
    end
end
end

```

#### 4.TG2-2S:

```

function [A1,B1,f1,A2,B2,f2,C2] = system_TG22S(xnode,a)
% [A1,B1,f1,A2,B2,f2,C2] = system_TG32S(xnode,a) TWO STEP
% The spatial discretization is performed using linear finite
% elements and the Galerkin formulation.
% xnode: nodal coordinates
% a : convection velocity
%
global dt
%alpha=1/9; %%ALPHA= 1/9 (TG3); ALPHA 1/12 ( TG4)
%dt_2 = dt*dt/2;
%dt2_alpha = dt^2*alpha; %%%%%%%%%%%%% ALPHA
% Gauss points and weights on the reference element [-1,1]
xipg = [-1/sqrt(3) 1/sqrt(3)]';
wpg = [1 1]';
% Shape functions on the reference element
N_mef = [(1-xipg)/2 (1+xipg)/2];
Nxi_mef = [-1/2 1/2; -1/2 1/2];
% Total number of nodes and elements
numnp = size(xnode,2);

```

```

numel = numnp-1;
% Number of Gauss points in each element
ngaus = size(wpg,1);
% Allocate storage
A1 = zeros(numnp,numnp);
B1 = zeros(numnp,numnp);
f1 = zeros(numnp,1);
A2 = zeros(numnp,numnp);
B2 = zeros(numnp,numnp);
f2 = zeros(numnp,1);
C2 = zeros(numnp,numnp);
% MATRICES COMPUTATION
% Loop on the elements
for i=1:numel
unos = ones (ngaus,1);
h = xnode(i+1)-xnode(i);
xm = (xnode(i)+xnode(i+1))/2;
weight = wpg*h/2;
isp = [i i+1];
% Loop on Gauss points (numerical quadrature)
for ig = 1:ngaus
N = N_mef(ig,:);
Nx = Nxi_mef(ig, :)*2/h;
w_ig = weight(ig);
x = xm + h/2*xipg(ig); % x-coordinate of Gauss point
% Matrices assembly
A1(isp,isp) = A1(isp,isp) + w_ig*(N'*N);
B1(isp,isp) = B1(isp,isp) - w_ig*((dt/2*N'*(a*Nx)));
f1(isp) = f1(isp) + w_ig*(N')*SourceTerm(x);
A2(isp,isp) = A2(isp,isp) + w_ig*(N'*N);
B2(isp,isp) = B2(isp,isp);
f2(isp) = f2(isp) + w_ig*(N')*SourceTerm(x);
C2(isp,isp) = C2(isp,isp) - w_ig*(dt*N'*(a*Nx));
end
end

```

### STEPS TO WRITE THE ENTIRE MATRIX:-

```

% Entire matrix (including boundary condition);
%Atot = [A Accd';Accd 0];
%[L,U] = lu(Atot);
if meth == 8 % 2-step method
Altot = [A1 Accd';Accd zeros(2)];
[L1,U1] = lu(Altot);
A2tot = [A2 Accd';Accd zeros(2)];
[L2,U2] = lu(A2tot);
else
Atot = [A Accd';Accd zeros(2)];
[L,U] = lu(Atot);
end

```

### SOLUTION STEPS:-

```

% SOLUTION AT EACH TIME STEP
for n = 1:nstep
if meth == 8 % 2-step method
btot = [B1*u(:,n)+ f1; bccd];
aux = U1\ (L1\btot);
u_m = u(:,n) + aux(1:numnp);
btot = [C2*u_m + f2; bccd];

```

```
aux = U2\ (L2\btot);  
u(:,n+1) = u(:,n) + aux(1:numnp);  
else  
btot = [B*u(:,n)+f; bccd];  
aux = U\ (L\btot);  
u(:,n+1) = u(:,n) + aux(1:numnp);  
end  
end
```

---- **END**----