Propagation of a steep front

Problem Statement:-

For the solution of one-dimensional transient pure convection problem

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

for the given problem, S=0

With the given initial

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

u (0, t) =1 , for the homogeneous Dirichlet inflow boundary condition

1. Courant Number:

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

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

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

Weighted Residual and Galerkin Formulation:-

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

At unit speed of discontinuous initial data, the given 1D problem considers the convection. At position *x* = 0.2 of the computational domain (0, 1), the discontinuity occurs over one element and is initially located. The given inlet condition is imposed. Uniform linear elements of size h=0.02 is employed as a mesh. In Figure 1, the results at time t = 0.6 are displayed 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,

Spurious oscillations over the whole computational domain being induced, can be observed that the Crank—Nicolson Scheme with Galerkin formulation. Because Crank—Nicolson is not a monotone scheme, residual oscillations remain at the front.

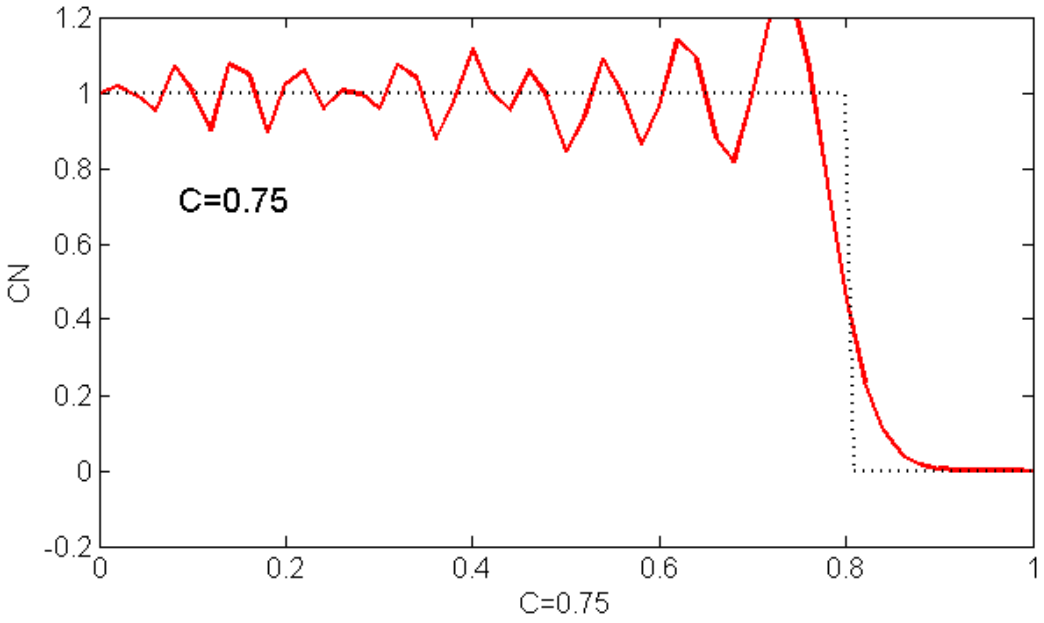

Figure 1: Using the Crank—Nicolson scheme with the Galerkin, propagation of a steep front Method at C =0.75. The graphs show the computed solutions at time t = 0.60, together with the exact solution.

3. Crank-Nicholson scheme in time and theleast-squares formulation in space

Crank-Nicholson scheme in time:

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

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 $L = \frac{1}{\Delta t} + \frac{1}{2}(\boldsymbol{a}\,.\boldsymbol{\nabla})$ is the spatial differential operator and f = $-\boldsymbol{a} * \boldsymbol{\nabla} u^n$.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.

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

At time t = 0.6, the results are displayed in Figure 2 together with the exact solution. by combining the Crank—Nicolson scheme (with linear elements) and the least-squares formulation of Carey and Jiang, they were obtained (for a Courant number C = 0.75). By the

Galerkin formulation over the whole computational domain, It is observed that Crank—Nicolson with least-squares succeeds in removing the spurious oscillations is induced

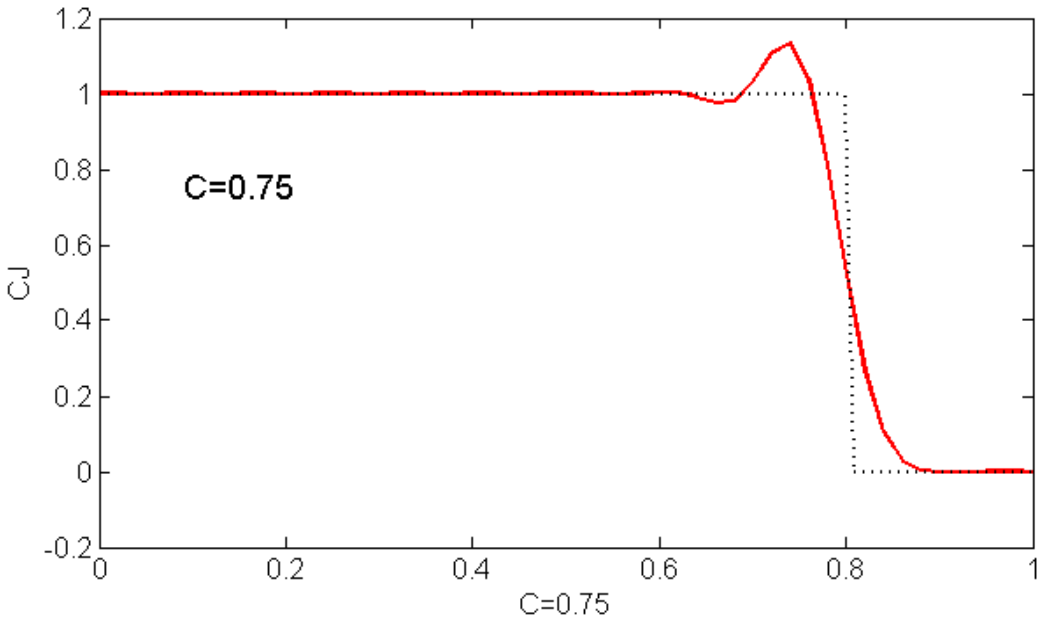

Figure 2: Using the Crank—Nicolson scheme with the least-squares method, propagation of a steep front. The Courant number is C = 0.75. The graphs show the computed solutions at time t = 0.60, together with the exact solution.

4. Second-order Lax-Wendroffmethod

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:-

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

Galerkin Formulation:-

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

$$(w, \frac{\Delta u}{\Delta t}) = (a w_x , u^n - \frac{\Delta t}{2} \boldsymbol{a}.\boldsymbol{\nabla} u^n)$$

In Figure 3, the results at time t = 0.6 are displayed 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. The number of time steps should be reduced to obtain good results so that the courant number will fall in the stability range of the TG2 scheme. In the Figure 4, the improved solutions obtained using the TG2 scheme is shown for C=0.3 is shown.

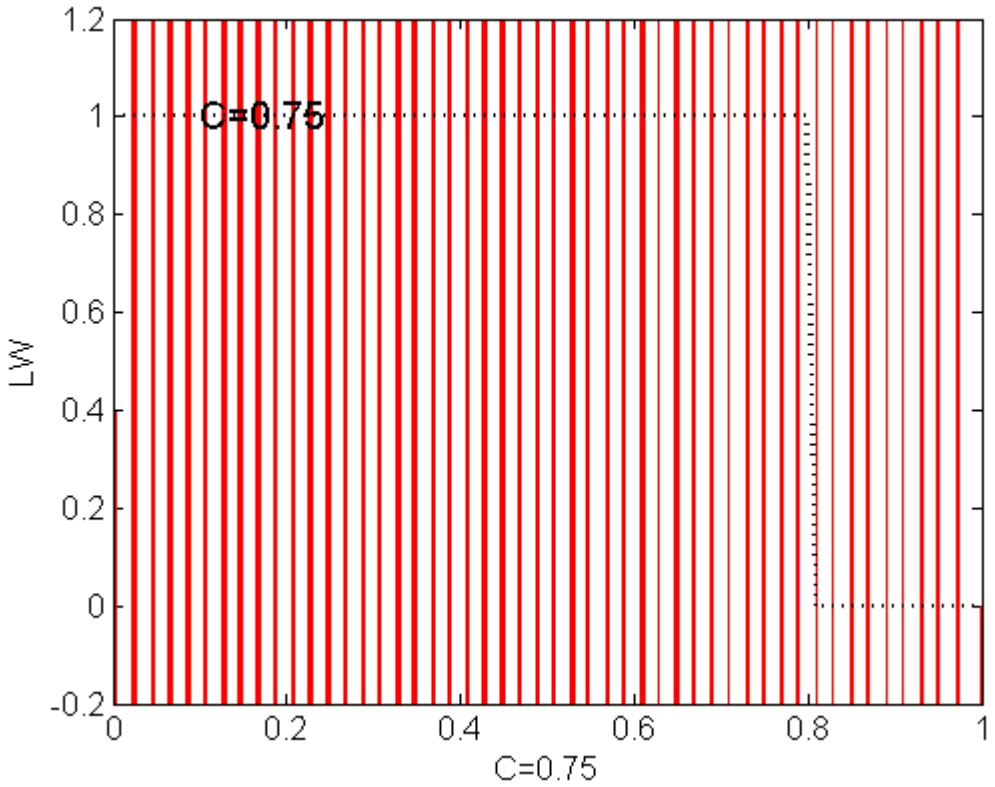

Figure 3: For the C= 0.75, solutions obtained using the TG2 method

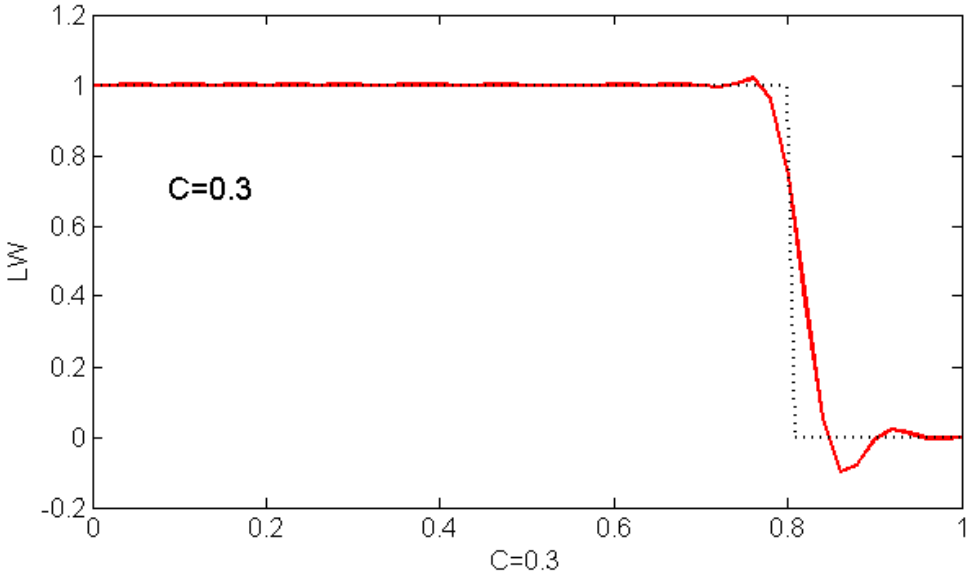

Figure 4: for the C= 0.3, solutions obtained using the TG2 method

5. TG2-2S

The explicit Taylor—Galerkin method TG2 of two-step versions 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:-

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

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

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

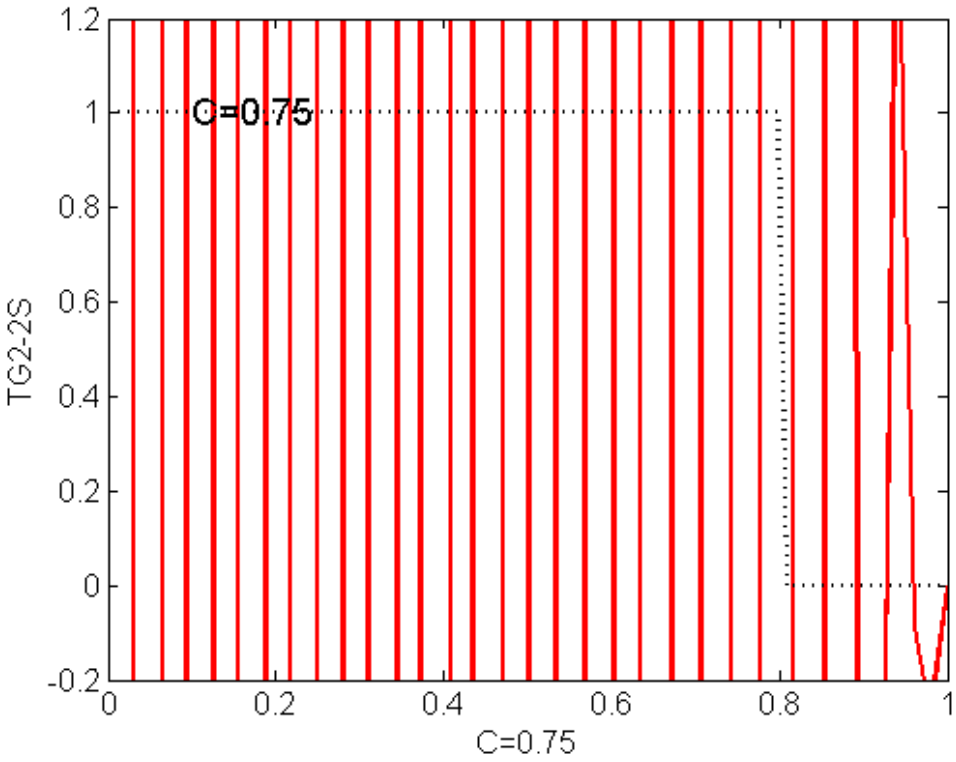

Figure 5: for the C= 0.75 solutions obtained using the TG2-2S method

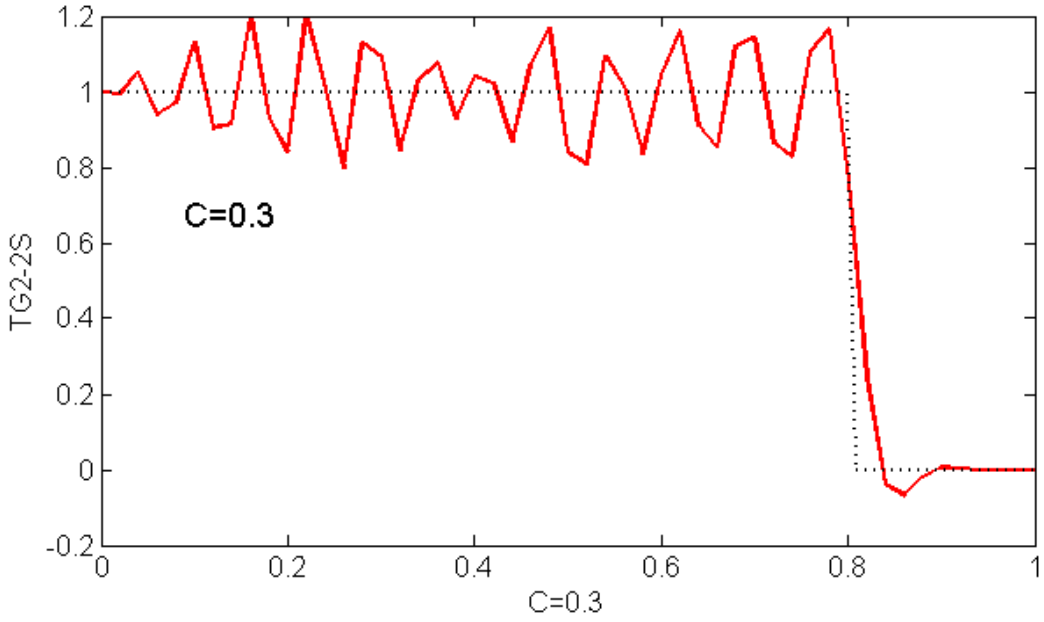

Figure 6: Solutions obtained using the TG2-2S (C=0.3)

CODE:-

Implementation of Initial Conditions:

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


globaldt

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
fori=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)
forig=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
```

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
%

globaldt

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
fori=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)
forig=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
```

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
%

globaldt

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
fori=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)
forig=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
```

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
%
globaldt
%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
fori=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)
forig = 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
A1tot = [A1 Accd';Accd zeros(2)];
[L1,U1] = lu(A1tot);
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----