UPC, CIMNE, ETSECCP

Finite Element in Fluids - Assignment 1

Rafael Pacheco
77128580N

May 22, 2017

1 Exercise 1

A domain $\Omega = (0,2) \times (0,3) \in \mathbb{R}^2$. The boundary Γ, with Dirichlet and Neumann boundary conditions such that $\Gamma = \Gamma_d \cup \Gamma_n$, is defined by the following closed set as

$$\begin{aligned} \Gamma_1 &= (0,0) \times (0,3/2) \\ \Gamma_2 &= (0,0) \times (3/2,3) \\ \Gamma_3 &= (0,2) \times (3,3) \\ \Gamma_4 &= (2,2) \times (0,3) \\ \Gamma_5 &= (0,2) \times (0,0) \end{aligned} \tag{1.1}$$

Consider the transient convection-diffusion-reaction problem with the unknown "ρ", the convective term "a", the reaction term "σ" and source term "s".

$$\begin{aligned} \rho_t + a \cdot \nabla\rho + \nabla\cdot(\mu\nabla\rho) + \sigma\rho = s \quad &\text{in} \quad \Omega \\ \rho = 1 \quad &\text{in} \quad \Gamma_2 \\ \rho = 0 \quad &\text{in} \quad \Gamma_4 \end{aligned} \tag{1.2}$$

To solve the problem you have to use

1. SUPG and GLS for the spatial discretization.

2. Padé approximations to recover a 1-stage 2nd order scheme and a 2-stage 4th order scheme as well as a 2-stage explicit scheme.

1.1 A) DISCRETIZATION

Time Discretization :

1.1.1 EXPLICIT PADÉ 2-STAGE , $R_{2,0}$

Using the $R_{2,0}$ a two-stage method can be derived from the factorization:

$$\rho(t^{n+1}) = \rho(t^n) + \Delta t \frac{\partial}{\partial t}\Big(\rho + \frac{\delta t}{2}\frac{\partial \rho}{\partial t}\Big)\Big|_{t=t^n} + \mathcal{O}(\Delta t^3) \tag{1.3}$$

Yielding the two-stage Lax-Wendroff method:

$$\begin{aligned} \rho^{n+1/2} &= \rho^n + \tfrac{\Delta}{2}\rho_t^n \\ \rho^{n+1} &= \rho^n + \Delta t \rho_t^{n+1/2} \end{aligned} \tag{1.4}$$

The explicit weak form is:

$$(\omega, \rho^{n+\beta_i}) = (\omega, \rho^n) + \beta_i \Delta t \Big[(\omega, s^{n+\beta_{i-1}}) + (\omega, h^{n+\beta_{i-1}})_{\Gamma_N} - c(a; \omega, \rho^{n+\beta_{i-1}}) - a(\omega, \rho^{n+\beta_{i-1}}) - (\omega, \sigma\rho^{n+\beta_{i-1}})\Big] \tag{1.5}$$

1.1.2 CRANK-NICOLSON, IMPLICIT PADÉ 1-STAGE 2ND ORDER, $R_{1,1}$

Using the definition of implicit Padé form $R_{n,n}$:

$$\begin{aligned} &\tfrac{\Delta\rho}{\Delta t} - W\Delta\rho_t = w\rho_t^n \\ &\tfrac{\Delta\rho}{\Delta t} + W\mathscr{L}(\Delta\rho) = w[s^n - \mathscr{L}(\rho^n)] + W\Delta s \\ &(w, \tfrac{\rho}{\Delta t}) + c(a; \omega, W\Delta\rho) + a(\omega, W\Delta\rho) + (\omega, \sigma W\Delta\rho) = \\ &(\omega, ws^n + W\Delta s) + (\omega, wh^n + W\Delta h) \\ &-[c(a; \omega, w\rho^n) + a(\omega, w\rho^n) + (\omega, \sigma w\rho^n)] \end{aligned} \tag{1.6}$$

$R_{1,1}$ is:

$$\begin{aligned} &\Delta\rho = \rho^{n+1} - \rho^n \quad && \Delta s = s^{n+1} - s^n \\ &W = 1/2 && w = 1 \end{aligned} \tag{1.7}$$

1.1.3 Implicit Padé 2-stage 4th order, $R_{2,2}$

$R_{2,2}$ is:

$$\begin{aligned} \Delta\rho &= \begin{bmatrix} \rho^{n+1/2} - \rho^n \\ \rho^{n+1} - \rho^{n+1/2} \end{bmatrix} & \Delta s &= \begin{bmatrix} s^{n+1/2} - s^n \\ s^{n+1} - s^{n+1/2} \end{bmatrix} \\ W &= \tfrac{1}{24}\begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix} & w &= \tfrac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{aligned} \tag{1.8}$$

Space Discretization :

Recall:

$$a(\omega,\rho) = \textstyle\int_\Omega \nabla\omega\cdot(\mu\nabla\rho)d\Omega \quad , \quad c(a;\omega,\rho) = \int_\Omega \omega(a\cdot\nabla\rho)d\Omega \tag{1.9}$$

The system to solve is:

$$M\rho_t + (C + K + \sigma M)\rho = f \tag{1.10}$$

And from equation (1.6) , we can rearrange the terms for a semi-discrete scheme and a multi-scale scheme respectively:

$$\begin{aligned} &(\omega,\rho_t + a\cdot\nabla\rho + \sigma\rho) + a(\omega,\rho) + \textstyle\sum_{e=1}^{n_{el}} \left(\mathscr{P}(\omega),\tau\mathscr{R}(\rho)\right)_\Omega = (\omega,s) + (\omega,h)_{\Gamma_N} \\ &\mathscr{R} = \rho_t + a\cdot\nabla\rho + \sigma\rho - \nabla\cdot(\mu\nabla\rho) - s \\ \\ &(\omega,\tfrac{\Delta\rho}{\Delta t}) - (\omega,W\Delta\rho_t) + \left(\tau\mathscr{P}(\omega),\mathscr{R}(\Delta\rho)\right)_\Omega = (\omega,w\rho_t^n) \\ &\mathscr{R} = \tfrac{\Delta\rho}{\Delta t} + W\mathscr{L}(\Delta\rho) - w[s^n - \mathscr{L}(\rho^n) - W\Delta s \\ &\mathscr{L} = a\cdot\nabla - \mu\nabla^2 + \sigma \end{aligned} \tag{1.11}$$

1.1.4 SUPG

$$\mathscr{P}(\omega) = W(a\cdot\nabla)\omega \tag{1.12}$$

1.1.5 GLS

$$\mathscr{P}(\omega) = \frac{\omega}{\Delta t} + W\mathscr{L}(\omega) \tag{1.13}$$

1.2 B) MODIFICATIONS

This are the new introduced subroutines:

FEM subroutines:

- CreateMatrix.m : create system resolution for steady case (K,f).
- CreateKMat.m: create the stiffnes matrix for unsteady case (K).
- FemMat: to solve the system from equation (1.10).

Scheme subroutines:

- SUPG.m: Introduced the SUPG scheme explained in section (1.1.4) together with equation (1.11).
- GLS.m: Introduced the GLS scheme explained in section (1.1.5) together with equation (1.11).
- PadeMat.m: Introduced the Padé scheme explained in section 1.1 Time discretization (1.1.1 , 1.1.2 , 1.1.3).

This together with some "cinput" entries to simulate a GUI to make the user able to select between steady/unsteady, the grid size, type of elements (for steady case) and the different coefficients. The subroutines are attached in the Appendix.

1.3 C)UNSTEADY PROBLEM

$$a = (1e-3, 0) \quad \nu = 1e-3 \quad \sigma = 1 \quad s = 0 \tag{1.14}$$

Consider all combinations.

The following plots represent $\rho(x, y)$.

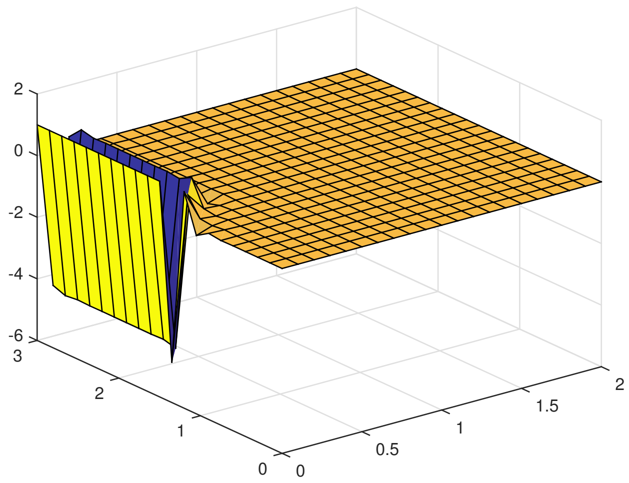

Figure 1.1: $R_{1,1}$ and GLS.

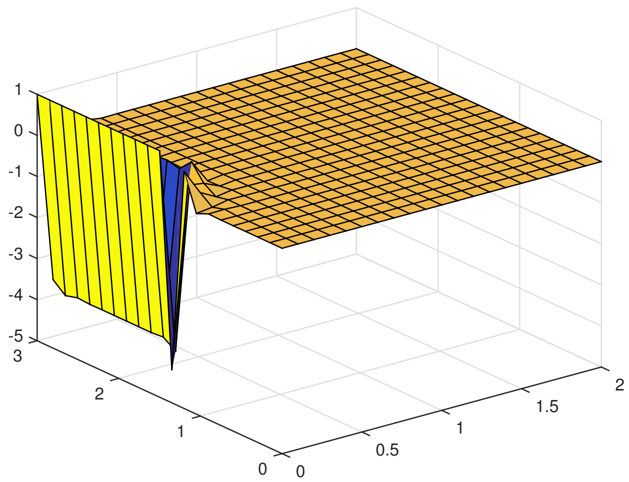

Figure 1.2: $R_{1,1}$ and SUPG.

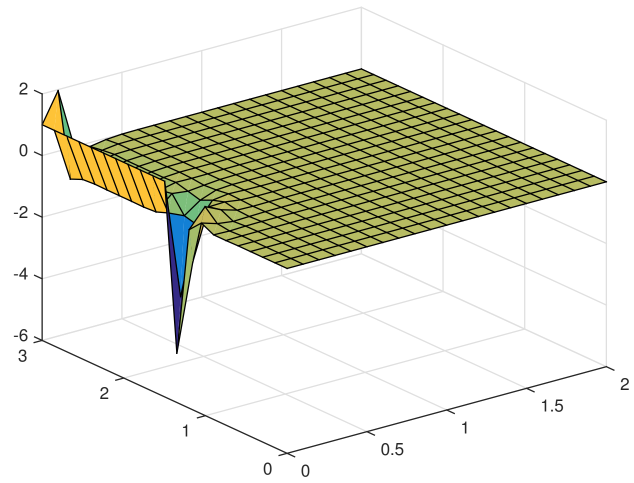

Figure 1.3: $R_{2,2}$ and GLS.

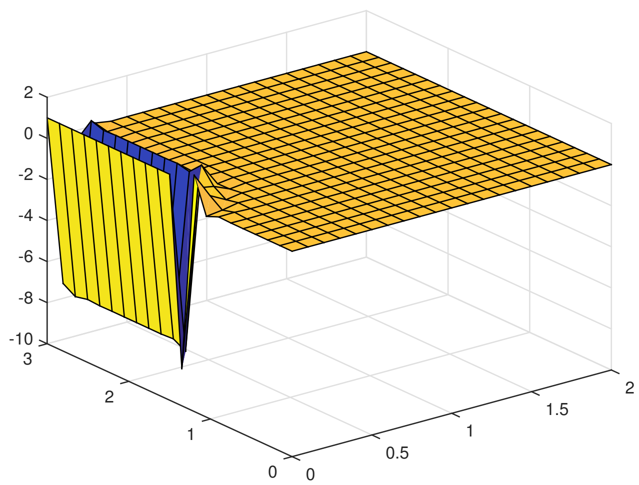

Figure 1.4: $R_{2,2}$ and SUPG.

For the explicit Padé $R_{2,0}$, it becomes unstable. It can be observed that method $R_{2,2}$ offers more accuracy for temporal domain and GLS for spatial domain.

1.4 D) STEADY PROBLEM

$$
\begin{array}{lllll}
Case: 1 & a=(-1,0) & \nu = 1e-3 & \sigma = 1e-1 & s=0 \\
Case: 2 & a=(-1e-3,0) & \nu = 1e-3 & \sigma = 1 & s=0 \\
Case: 3 & a=(-1e-3,0) & \nu = 1e-3 & \sigma = 0 & s=1
\end{array}
\tag{1.15}
$$

Compare the results for the different spatial discretizations and comment the results. Note all the results are for linear elements.

1.4.1 CASE 1

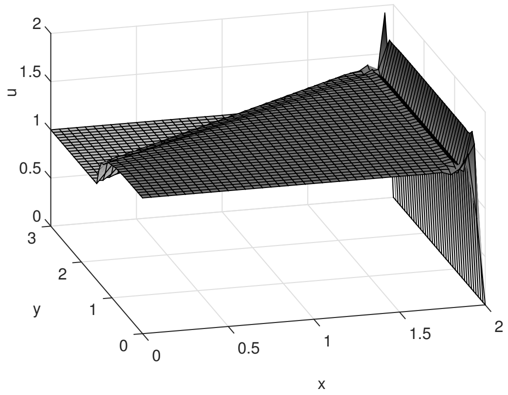

Figure 1.5: Case 1 and GLS.

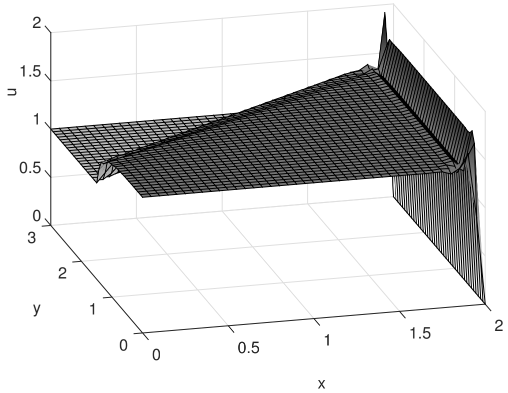

Figure 1.6: Case 1 and SUPG.

1.4.2 Case 2

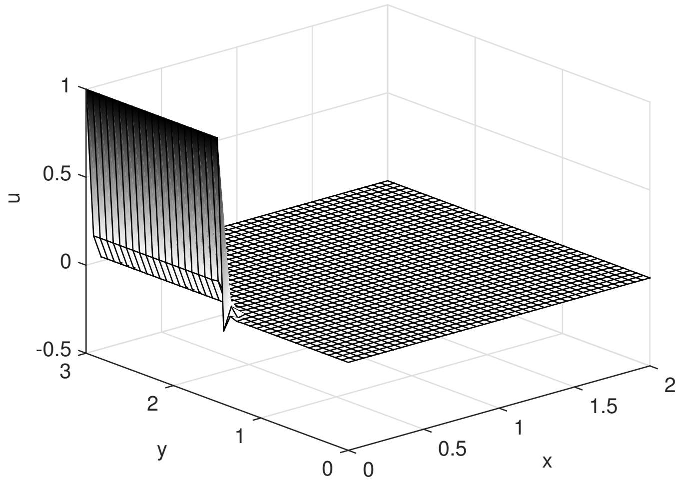

Figure 1.7: Case 2 and GLS.

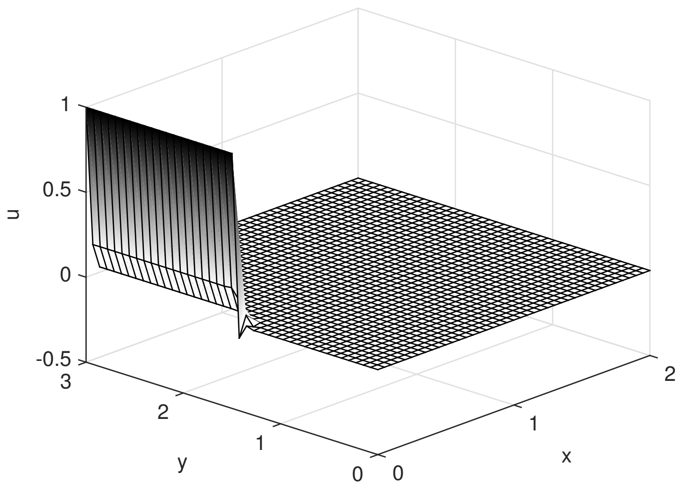

Figure 1.8: Case 2 and SUPG.

1.4.3 Case 3

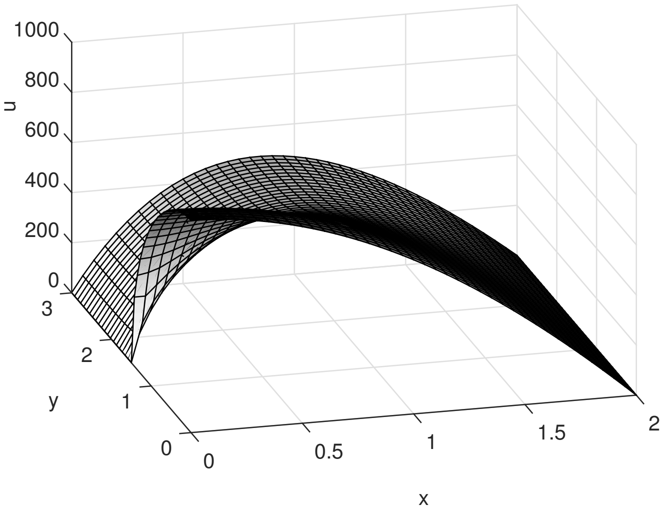

Figure 1.9: Case 3 and GLS.

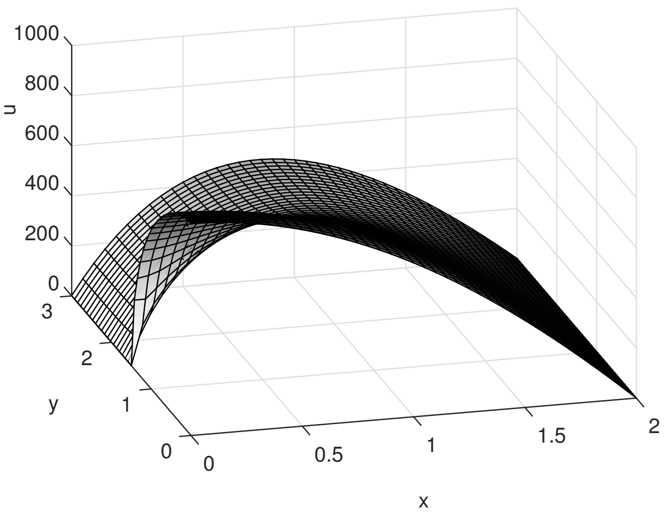

Figure 1.10: Case 3 and SUPG.

In case 1, the difference is relatively small, and therefore it cannot be appreciated in the graphs. However, this difference should be of the order $(1+\tau\sigma)$.

For cases 2 and 3, the difference on the SUPG and GLS scheme is not relatively important since the convective term is not dominant. Therefore using a Galerkin scheme should be enough, since these 2 last cases do not require stabilization.

APPENDIX

```
function [K,f] = CreateMatrix(X,T,pospg,wpg,N,Nxi,Neta,nu,sigma,a,method,h,Conv,s)
% [K,f] = CreateMatrix(X,T,pospg,wpg,N,Nxi,Neta)
%
% Stiffness matrix K and r.h.s vector f obtained by discretizing
% a convection-diffusion-reaction equation in 2D
%   X:            nodal coordinates
%   T:            connectivities (elements)
%   pospg, wpg:   Gauss points an weights in the reference element
%   N,Nxi,Neta:   shape functions on the Gauss points
%
ax = Conv(1);
ay = Conv(2);


% Total number of elements and number of nodes in aech one
[numel,nen] = size(T);
% Total number of nodes
numnp = size(X,1);

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

if method == 1
    disp('Galerkin formulation');
    tau = 0;
elseif method == 2
    disp ('SUPG formulation');
    Pe = a*h/(2*nu);
    % alpha = coth(Pe)-1/Pe;
    % tau_p = alpha*h/(2*a);
    tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
    disp(strcat('Recommended value for the stabilization parameter = ',
        num2str(tau_p)));
    tau = input('Stabilization parameter to be used (press return for
        using the recommended one)= ');
    if isempty(tau)
        tau = tau_p;
    end
elseif method == 3
    disp('GLS formulation');
```

```
    Pe = a*h/(2*nu);
    % alpha = coth(Pe)-1/Pe;
    % tau_p = alpha*h/(2*a);
    tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
    disp(strcat('Recommended value for the stabilization parameter = ',
        num2str(tau_p)));
    tau = input('Stabilization parameter to be used (press return for
        using the recommended one)= ');
    if isempty(tau)
        tau = tau_p;
    end
elseif method == 4
    disp('SGS formulation');
    Pe = a*h/(2*nu);
    % alpha = coth(Pe)-1/Pe;
    % tau_p = alpha*h/(2*a);
    tau_p = h*(1 + 9/Pe^2 +(h*sigma/(2*a))^2)^(-1/2)/(2*a);
    disp(strcat('Recommended value for the stabilization parameter =',
        num2str(tau_p)));
    tau = input('Stabilization parameter to be used (press return for
        using the recommended one)= ');
    if isempty(tau)
        tau = tau_p;
    end
else
   error ('Unavailable method')
end

% Loop on the elements
for ielem = 1:numel
    % Te: global number of the nodes in the current element
    % Xe: coordenates of the nodes in the current element
    Te = T(ielem,:);
    Xe = X(Te,:);
    [Ke,fe] = MatEl(Xe,nen,pospg,wpg,N,Nxi,Neta,tau,nu,sigma,a,Conv,h,s,
        method);
    % Assemble the element matrices
    K(Te,Te)=K(Te,Te)+Ke;
    f(Te)=f(Te)+fe;
    clear Ke; clear fe;
end
```

```
function K2 = CreateKMat (X,T,Conv,referenceElement)
% K2 = CreK2Mat (X,T,Conv,pospg,wpg,N,Nxi,Neta);
```

```
% Computation of the matrix K2 obtained by discretizing (aůgrad(w),
    aůgrad(u))
%

% reference element information
nen = referenceElement.nen;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;

% Total number of elements and number of nodes in each element
nElem = size(T,1);
% Total number of nodes
nPt = size(X,1);

% Allocation
K2 = zeros(nPt);

% Loop on the elements
for ielem = 1:nElem
    % Te: global number of nodes on the current element
    Te = T(ielem,:);
    % Xe: coordinates of the nodes in Te
    Xe = X(Te,:);
    % Conve: velocity field on Te
    Conve = Conv(Te,:);
    % Element Matrix
    K2e = zeros(nen,nen);
    % Loop on Gauss points (numerical quadrature)
    for ig = 1:ngaus
        % Shape functions on Gauss point igaus
        N_ig    = N(ig,:);
        Nxi_ig  = Nxi(ig,:);
        Neta_ig = Neta(ig,:);
        % Jacobian matrix on the Gauss point
        Jacob = [Nxi_ig*(Xe(:,1))     Nxi_ig*(Xe(:,2))
                 Neta_ig*(Xe(:,1))    Neta_ig*(Xe(:,2))];
        %
        dvolu = wgp(ig)*det(Jacob);
        % Derivatives of shape functions in global coordinates
        res = Jacob\[Nxi_ig;Neta_ig];
        Nx = res(1,:);
```

```
            Ny = res(2,:);
            % Contribution at the element matrix
            a = N_ig*Conve; ax = a(1); ay = a(2);
            K2e = K2e + (ax*Nx+ay*Ny)'*(ax*Nx+ay*Ny)*dvolu;
        end
        % Assembly of the element matrix
        K2(Te,Te) = K2(Te,Te) + K2e;
        clear K2e;
end
```

```
function [M,K,C] = FEMmat(X,T,Conv,referenceElement)
% This function compute the different spatial matrices which are
    involved
% into the unsteady convection−diffusion−reaction problem.

    % C = convection matrix C obtained by discretizing (w, aůgrad(u))
    % K = stiffness matrix K obtained by discretizing (grad(w), grad(u))
    % M = mass matrix M obtained by discretizing (w, u)

% reference element information
nen = referenceElement.nen;
ngaus = referenceElement.ngaus;
wgp = referenceElement.GaussWeights;
N = referenceElement.N;
Nxi = referenceElement.Nxi;
Neta = referenceElement.Neta;

% Total number of nodes
nPt = size(X,1);

% Total number of elements and number of nodes in each element
nElem = size(T,1);

% Allocation
M = zeros(nPt);
K = zeros(nPt);
C = zeros(nPt);


% Loop on elements
for ielem=1:nElem
    % Te: global number of nodes on the current element
    Te = T(ielem,:);
    % Xe: coordinates of the nodes in Te
```

```
    Xe = X(Te,:);
    % Conve: velocity field on Te
    Conve = Conv(Te,:);
    % Element Matrix
    [Me,Ke,Ce] = EleMat(Xe,Conve,nen,ngaus,wgp,N,Nxi,Neta);
    % Assembly
    M(Te,Te) = M(Te,Te) + Me;
    K(Te,Te) = K(Te,Te) + Ke;
    C(Te,Te) = C(Te,Te) + Ce;

end



function [Me,Ke,Ce] = EleMat(Xe,Conve,nen,ngaus,wgp,N,Nxi,Neta)
%

Me = zeros(nen);
Ke = zeros(nen);
Ce = zeros(nen);


% Loop on Gauss points
for ig = 1:ngaus
    % Shape functions on Gauss point igaus
    N_ig = N(ig,:);
    Nxi_ig = Nxi(ig,:);
    Neta_ig = Neta(ig,:);
    % Jacobian matrix on the Gauss point
    Jacob = [Nxi_ig*(Xe(:,1))   Nxi_ig*(Xe(:,2))
             Neta_ig*(Xe(:,1))  Neta_ig*(Xe(:,2))];
    % Differential of volume
    dvolu = wgp(ig)*det(Jacob);
    % Derivatives of shape functions in global coordinates
    res = Jacob\[Nxi_ig;Neta_ig];
    Nx = res(1,:);
    Ny = res(2,:);
    % Contribution at the element matrix
    a = N_ig*Conve; ax = a(1); ay = a(2);

    Me = Me + N_ig'*N_ig*dvolu;
    aGradN = ax*Nx + ay*Ny;
    Ke = Ke + (Nx+Ny)'*(Nx+Ny)*dvolu;
    Ce = Ce + N_ig'*aGradN*dvolu;
```

```
end
```

```
function Sol = SUPG(X,T,Conv,nu,sigma,f,c,Accd1,bccd1,W,w,dt,nstep,
    referenceElement,M,C,K,nodesDir0,nodesDir1,tau)
% Sol = SUPG(X,IEN,Conv,nu,f,c,Accd1,bccd1, T,s,beta,dt,nstep,tau)
% This function computes solution for a transient convection-diffusion
    equation
% at different time instants.
% SUPG method is used to perform spatial discretization.
%
% Input:
%   X: nodal coordinates
%   T: connectivities
%   Conv: velocity field
%   nu: diffusion
%   sigma: reaction term
%   f: source term
%   c: initial condition
%   Accd1, bccd1: matrices to impose boundary conditions using Lagrange
    multipliers
%   W,w: matrices for the time-integration scheme
%   dt: time-step
%   nstep: number of time steps
%   referenceElement: Information about the element of reference
%   M: mass matrix
%   C: convection matrix
%   K: stiffness matrix
%   tau: stabilization parameter
%   nodesDir0: nodes where the value of u is 0
%   nodesDir1: nodes where the value of u is 1

% Number of points used in the discretization
npoin = size(X,1);

% COMPUTATION OF MATRIX K2

K2 = CreK2Mat (X,T,Conv,referenceElement);

% Integration matrix
[n,m] = size(W);
Id = eye(n,m);
tauW = tau*W;
tauW_W = tauW'*W;
tauW_w = tauW'*w;
```

```
if size(tau) == [1,1]
    tau = tau*ones(size(W));
end

% Computation of the matrix necessary to obtain solution at each time-
    step: A du = F
disp(' ')
disp('Computation of total matrices for the time-integration scheme')
Kt  = C + nu*K+sigma*M;
A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [ row, Id(i,j)*M  + dt*W(i,j)*Kt + ...
                       tauW(j,i)*C' + dt*tauW_W(i,j)*(K2+sigma*C')  ];
    end
     A = [A; row];
end

nccd = size(Accd1,1);
Accd = []; bccd = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end

nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];

% Factorization of matrix Atot
disp(' ')
disp('Factorization of the matrices for the time-integration scheme')
[L,U] = lu(Atot);
L = sparse(L);
U = sparse(U);

% Initial condition
Sol = c;
% Loop to compute the transient solution
```

```
disp(' ')
disp('Transient analysis: computation of the solution at each time step'
    )
for i = 1:nstep
    aux1 = -dt*(Kt*c);
    aux2 = -dt*((K2+sigma*C')*c);
    F = [];
        for j=1:m
            F = [F; w(j)*aux1 + tauW_w(j)*aux2];
        end
    F = [F;bccd];
    dc  = U\(L\F);
    dc = reshape(dc(1:n*npoin),npoin,n);
    for k =1:size(dc,1)
        if any(nodesDir1 == k);
            c(k)=1;
        elseif any(nodesDir0 == k);
            c(k)=0;
        else
            c(k) = c(k) + sum(dc(k,:),2);
        end
    end
    Sol = [Sol c];
end
```

```
function Sol = GLS(X,T,Conv,nu,sigma,f,c,Accd1,bccd1,W,w,dt,nstep,
    referenceElement,M,C,K,nodesDir0,nodesDir1,tau)
% Sol = SUPG(X,IEN,Conv,nu,f,c,Accd1,bccd1, T,s,beta,dt,nstep,tau)
% This function computes solution for a transient convection-diffusion
    equation
% at different time instants.
% GLS method is used to perform spatial discretization.
%
% Input:
%   X: nodal coordinates
%   T: connectivities
%   Conv: velocity field
%   nu: diffusion
%   sigma: reaction term
%   f: source term
%   c: initial condition
%   Accd1, bccd1: matrices to impose boundary conditions using Lagrange
    multipliers
%   W,w: matrices for the time-integration scheme
```

```
%    dt: time-step
%    nstep: number of time steps
%    referenceElement: Information about the element of reference
%   M: mass matrix
%    C: convection matrix
%    K: stiffness matrix
%    tau: stabilization parameter
%    nodesDir0: nodes where the value of u is 0
%    nodesDir1: nodes where the value of u is 1

% Number of points used in the discretization
npoin = size(X,1);

% COMPUTATION OF MATRIX K2

K2 = CreK2Mat (X,T,Conv,referenceElement);

% Integration matrix
 [n,m] = size(W);
Id = eye(n,m);
tauW = tau*W;
tauw = tau*w;
tauW_W = tauW'*W;
tauW_w = tauW'*w;

 if size(tau) == [1,1]
     tau = tau*ones(size(T));
end

% Computation of the matrix necessary to obtain solution at each time-
      step: A du = F
disp(' ')
disp('Computation of total matrices for the time-integration scheme')
Kt   = C + nu*K+sigma*M;
A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [ row, Id(i,j)*M  + dt*W(i,j)*Kt + ...
              tau(i,j)*(1/dt)*M + tauW(i,j)*(C+sigma*M) +...
              tauW(j,i)*(C'+sigma*M)+dt*tauW_W(i,j)*(K2+sigma*C') +...
              dt*tauW_W(i,j)*sigma*(C+sigma*M)  ];
    end
     A = [A; row];
```

```
end

nccd = size(Accd1,1);
Accd = []; bccd = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end

nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];

% Factorization of matrix Atot
disp(' ')
disp('Factorization of the matrices for the time-integration scheme')
[L,U] = lu(Atot);
L = sparse(L);
U = sparse(U);

% Initial condition
Sol = c;
% Loop to compute the transient solution
disp(' ')
disp('Transient analysis: computation of the solution at each time step'
    )
for i = 1:nstep
    aux1 = -dt*Kt*c;
    aux2 = -(C+sigma*M)*c;
    aux3 = dt*(-(K2+sigma*C'+sigma*(C +sigma*M))*c);
    F = [];
        for j=1:n
            F = [F; w(j)*aux1 + tauw(j)*aux2 + tauW_w(j)*aux3];
        end
    F = [F;bccd];
    dc  = U\(L\F);
    dc = reshape(dc(1:n*npoin),npoin,n);
    for k =1:size(dc,1)
        if any(nodesDir1 == k);
            c(k)=1;
        elseif any(nodesDir0 == k);
```

```
                c(k)=0;
            else
                c(k) = c(k) + sum(dc(k,:) ,2);
            end
        end
        Sol = [Sol c];
end
```

```
function [W,w,beta]=Pade(d)

% Compute the matrices needed to perform the time integration Pad scheme

if d == 0
    W = 1/2;
    w = 1;
    beta = [0,1];
elseif d == 1
    W = (1/24)*[7 −1; 13 5];
    w = [1/2; 1/2];
    beta = [0,1/2,1];
elseif d == 2
    W = [0 0; 1 0];
    w = [1/2; 1/2];
    beta = [0,1/2,1];
else
    error('Unavailable time integration scheme')
end


end
```