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

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

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

$$\rho_{t} + a \cdot \nabla \rho + \nabla \cdot (\mu \nabla \rho) + \sigma \rho = s \quad \text{in} \quad \Omega$$

$$\rho = 1 \qquad \text{in} \quad \Gamma_{2}$$

$$\rho = 0 \qquad \text{in} \quad \Gamma_{4}$$
(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.







Rafael Pacheco 77128580N

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:

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

Yielding the two-stage Lax-Wendroff method:

$$\rho^{n+1/2} = \rho^n + \frac{\Delta}{2} \rho_t^n$$

$$\rho^{n+1} = \rho^n + \Delta t \rho_t^{n+1/2}$$
(1.4)

The explicit weak form is:

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

1.1.2 Crank-Nicolson, Implicit Padé 1-stage 2nd order, $R_{1,1}$

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

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

*R*_{1,1} is:







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

*R*_{2,2} is:

$$\Delta \rho = \begin{bmatrix} \rho^{n+1/2} - \rho^n \\ \rho^{n+1} - \rho^{n+1/2} \end{bmatrix} \quad \Delta s = \begin{bmatrix} s^{n+1/2} - s^n \\ s^{n+1} - s^{n+1/2} \end{bmatrix}$$

$$W = \frac{1}{24} \begin{bmatrix} 7 & -1 \\ 13 & 5 \end{bmatrix} \qquad W = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
(1.8)

Space Discretization :

Recall:

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

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:

$$\begin{aligned} (\omega, \rho_t + a \cdot \nabla \rho + \sigma \rho) + a(\omega, \rho) + \sum_{e=1}^{n_{el}} \left(\mathscr{P}(\omega), \tau \mathscr{R}(\rho) \right)_{\Omega} &= (\omega, s) + (\omega, h)_{\Gamma_N} \\ \mathscr{R} &= \rho_t + a \cdot \nabla \rho + \sigma \rho - \nabla \cdot (\mu \nabla \rho) - s \end{aligned}$$

$$(\omega, \frac{\Delta \rho}{\Delta t}) - (\omega, W \Delta \rho_t) + \left(\tau \mathscr{P}(\omega), \mathscr{R}(\Delta \rho) \right)_{\Omega} &= (\omega, w \rho_t^n) \\ \mathscr{R} &= \frac{\Delta \rho}{\Delta t} + W \mathscr{L}(\Delta \rho) - w [s^n - \mathscr{L}(\rho^n) - W \Delta s \\ \mathscr{L} &= a \cdot \nabla - \mu \nabla^2 + \sigma \end{aligned}$$

$$(1.11)$$

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.







Finite Element in Fluids

- Assignment 1 -

Rafael Pacheco 77128580N

1.3 C) UNSTEADY PROBLEM

$$a = (1e - 3, 0)$$
 $v = 1e - 3$ $\sigma = 1$ $s = 0$ (1.14)

Consider all combinations.

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







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



Universitat Politècnica De Catalunya BARCELONATECH



Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona



Centre Internacional de Mètodes Numèrics en Enginyeria



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





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.



Universitat Politècnica De Catalunya BARCELONATECH



Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona



Rafael Pacheco 77128580N

1.4 D)STEADY PROBLEM

$$Case: 1 \quad a = (-1,0) \qquad v = 1e-3 \quad \sigma = 1e-1 \quad s = 0$$

$$Case: 2 \quad a = (-1e-3,0) \quad v = 1e-3 \quad \sigma = 1 \qquad s = 0$$

$$Case: 3 \quad a = (-1e-3,0) \quad v = 1e-3 \quad \sigma = 0 \qquad s = 1$$
(1.15)

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









Figure 1.6: Case 1 and SUPG.



Universitat Politècnica De Catalunya BARCELONATECH









Figure 1.7: Case 2 and GLS.



Figure 1.8: Case 2 and SUPG.



Universitat Politècnica De Catalunya BARCELONATECH



Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona







Figure 1.9: Case 3 and GLS.



Figure 1.10: Case 3 and SUPG.



Universitat Politècnica De Catalunya BARCELONATECH



Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona



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.



Universitat Politècnica De Catalunya BARCELONATECH



Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona



Centre Internacional de Mètodes Numèrics en Enginyeria Finite Element in Fluids

- Assignment 1 -

Rafael Pacheco 77128580N

APPENDIX

```
1 function [K, f] = CreateMatrix (X, T, pospg, wpg, N, Nxi, Neta, nu, sigma, a, method
       , h, Conv, s)
 2 \% [K, f] = CreateMatrix(X, T, pospg, wpg, N, Nxi, Neta)
 3 %
 4 % Stiffness matrix K and r.h.s vector f obtained by discretizing
 5 % a convection-diffusion-reaction equation in 2D
       X:
                    nodal coordinates
 6 %
       T:
                    connectivities (elements)
 7 %
       pospg, wpg: Gauss points an weights in the reference element
 8 %
       N, Nxi, Neta: shape functions on the Gauss points
 9 %
10 %
11 ax = Conv(1);
12 \text{ ay} = \text{Conv}(2);
13
14
15 % Total number of elements and number of nodes in aech one
16 [numel, nen] = size(T);
17 % Total number of nodes
18 nump = size(X, 1);
19
20 % Allocate storage
21 K = zeros(numnp, numnp);
22 f = zeros(numnp, 1);
23
24 if method == 1
       disp('Galerkin formulation');
25
       tau = 0;
26
27 elseif method == 2
       disp ('SUPG formulation');
28
       Pe = a * h / (2 * nu);
29
       \% alpha = coth(Pe) - 1/Pe;
30
       \% tau_p = alpha*h/(2*a);
31
       tau_p = h*(1 + 9/Pe^2 + (h*sigma/(2*a))^2)^{(-1/2)/(2*a)};
32
       disp(strcat('Recommended value for the stabilization parameter = ',
33
           num2str(tau_p)));
       tau = input('Stabilization parameter to be used (press return for
34
           using the recommended one) = ');
       if isempty(tau)
35
           tau = tau_p;
36
       end
37
38 elseif method == 3
39
       disp('GLS formulation');
```





```
Pe = a * h / (2 * nu);
40
      \% alpha = coth(Pe)-1/Pe;
41
42
      \% tau_p = alpha*h/(2*a);
       tau_p = h*(1 + 9/Pe^2 + (h*sigma/(2*a))^2)^{(-1/2)}/(2*a);
43
       disp(strcat('Recommended value for the stabilization parameter = ',
44
          num2str(tau_p)));
       tau = input('Stabilization parameter to be used (press return for
45
           using the recommended one) = ');
       if isempty(tau)
46
           tau = tau p;
47
48
       end
49 elseif method == 4
       disp('SGS formulation');
50
       Pe = a * h / (2 * nu);
51
       \% alpha = coth(Pe) - 1/Pe;
52
53
      \% tau_p = alpha*h/(2*a);
       tau_p = h*(1 + 9/Pe^2 + (h*sigma/(2*a))^2)^{(-1/2)}/(2*a);
54
       disp(strcat('Recommended value for the stabilization parameter =',
55
          num2str(tau_p)));
       tau = input('Stabilization parameter to be used (press return for
56
           using the recommended one) = ');
       if isempty(tau)
57
           tau = tau_p;
58
59
       end
60 else
      error ('Unavailable method')
61
62 end
63
64 % Loop on the elements
  for ielem = 1:numel
65
      % Te: global number of the nodes in the current element
66
      % Xe: coordenates of the nodes in the current element
67
      Te = T(ielem, :);
68
      Xe = X(Te,:);
69
       [Ke, fe] = MatEl (Xe, nen, pospg, wpg, N, Nxi, Neta, tau, nu, sigma, a, Conv, h, s,
70
          method);
      % Assemble the element matrices
71
      K(Te, Te) = K(Te, Te) + Ke;
72
       f(Te) = f(Te) + fe;
73
       clear Ke; clear fe;
74
75 end
```

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







```
3 % Computation of the matrix K2 obtained by discretizing (augrad(w),
      a \hat{u} grad(u)
4 %
5
6 % reference element information
7 nen = referenceElement.nen;
8 ngaus = referenceElement.ngaus;
9 wgp = referenceElement.GaussWeights;
10 \text{ N} = \text{referenceElement.N};
11 Nxi = referenceElement.Nxi;
12 Neta = referenceElement.Neta;
13
14 % Total number of elements and number of nodes in each element
15 nElem = size(T, 1);
16 % Total number of nodes
17 nPt = size(X, 1);
18
19 % Allocation
20 K2 = zeros(nPt);
21
22 % Loop on the elements
23 for ielem = 1:nElem
24
      % Te: global number of nodes on the current element
      Te = T(ielem, :);
25
      % Xe: coordinates of the nodes in Te
26
27
      Xe = X(Te,:);
      % Conve: velocity field on Te
28
      Conve = Conv(Te,:);
29
30
      % Element Matrix
       K2e = zeros(nen, nen);
31
      % Loop on Gauss points (numerical quadrature)
32
       for ig = 1:ngaus
33
           % Shape functions on Gauss point igaus
34
           N_ig
                   = N(ig, :);
35
           Nxi_i = Nxi(ig,:);
36
           Neta_ig = Neta(ig,:);
37
           % Jacobian matrix on the Gauss point
38
           Jacob = [Nxi_ig * (Xe(:,1))]
                                              Nxi_i (Xe(:,2))
39
                    Neta_ig * (Xe(:,1))
                                              Neta_ig * (Xe(:,2))];
40
           %
41
           dvolu = wgp(ig) * det(Jacob);
42
           % Derivatives of shape functions in global coordinates
43
           res = Jacob\[Nxi_ig;Neta_ig];
44
           Nx = res(1,:);
45
```





```
Ny = res(2,:);
46
           % Contribution at the element matrix
47
           a = N_{ig}*Conve; ax = a(1); ay = a(2);
48
           K2e = K2e + (ax*Nx+ay*Ny)'*(ax*Nx+ay*Ny)*dvolu;
49
       end
50
51
      % Assembly of the element matrix
      K2(Te,Te) = K2(Te,Te) + K2e;
52
       clear K2e;
53
54 end
1 function [M,K,C] = FEMmat(X,T,Conv,referenceElement)
2 % This function compute the different spatial matrices which are
      involved
3 % into the unsteady convection-diffusion-reaction problem.
4
      \% C = convection matrix C obtained by discretizing (w, augrad(u))
5
      \% K = stiffness matrix K obtained by discretizing (grad(w), grad(u))
6
      \% M = mass matrix M obtained by discretizing (w, u)
7
8
9 % reference element information
10 nen = referenceElement.nen;
11 ngaus = referenceElement.ngaus;
12 wgp = referenceElement.GaussWeights;
13 N = referenceElement.N;
14 Nxi = referenceElement.Nxi;
15 Neta = referenceElement.Neta;
16
17 % Total number of nodes
18 nPt = size(X, 1);
19
20 % Total number of elements and number of nodes in each element
21 nElem = size(T, 1);
22
23 % Allocation
24 M = zeros(nPt);
25 K = zeros(nPt);
26 \text{ C} = \mathbf{zeros}(n\text{Pt});
27
28
29 % Loop on elements
30 for ielem=1:nElem
       % Te: global number of nodes on the current element
31
       Te = T(ielem, :);
32
      % Xe: coordinates of the nodes in Te
33
```





Finite Element in Fluids

- Assignment 1 -

```
Xe = X(Te,:);
34
       % Conve: velocity field on Te
35
       Conve = Conv(Te,:);
36
       % Element Matrix
37
       [Me, Ke, Ce] = EleMat(Xe, Conve, nen, ngaus, wgp, N, Nxi, Neta);
38
39
       % Assembly
      M(Te,Te) = M(Te,Te) + Me;
40
       K(Te,Te) = K(Te,Te) + Ke;
41
       C(Te,Te) = C(Te,Te) + Ce;
42
43
44 end
45
46
47
48 function [Me, Ke, Ce] = EleMat(Xe, Conve, nen, ngaus, wgp, N, Nxi, Neta)
49 %
50
51 Me = zeros(nen);
52 Ke = zeros(nen);
53 Ce = zeros(nen);
54
55
56 % Loop on Gauss points
57 for ig = 1:ngaus
       % Shape functions on Gauss point igaus
58
       N_{ig} = N(ig, :);
59
60
       Nxi_i = Nxi(ig,:);
       Neta_ig = Neta(ig,:);
61
62
       % Jacobian matrix on the Gauss point
       Jacob = [Nxi_ig * (Xe(:,1))]
                                      Nxi_i (Xe(:,2))
63
                 Neta_ig * (Xe(:,1)) Neta_ig * (Xe(:,2)) ];
64
       % Differential of volume
65
       dvolu = wgp(ig)*det(Jacob);
66
       % Derivatives of shape functions in global coordinates
67
       res = Jacob \ [Nxi_ig; Neta_ig];
68
       Nx = res(1,:);
69
       Nv = res(2,:);
70
       % Contribution at the element matrix
71
       a = N_{ig}*Conve; ax = a(1); ay = a(2);
72
73
      Me = Me + N_{ig'} * N_{ig} * dvolu;
74
75
       aGradN = ax*Nx + ay*Ny;
       Ke = Ke + (Nx+Ny) '*(Nx+Ny)*dvolu;
76
77
       Ce = Ce + N_ig'*aGradN*dvolu;
```





Rafael Pacheco 77128580N

78 end

1	fu	Inction Sol = SUPG(X, T, Conv, nu, sigma, f, c, Accdl, bccdl, W, w, dt, nstep, referenceElement MC K, nodesDirl, nodesDirl, tau)
2	0%	Sol - SUPC(Y IEN Conv. nu. f. c. Accd1, hecd1, T s. heta, dt. nston, tau)
2	70 %	This function computes solution for a transient convection-diffusion
5	70	equation
4	%	at different time instants
4 5	70 %	SUPG method is used to perform spatial discretization
6	%	solo memou is used to perform sputtut discretization.
7	%	Innut
' 8	%	X: nodal coordinates
9	%	T: connectivities
10	%	Conv: velocity field
11	%	nu: diffusion
12	%	sigma: reaction term
13	%	f: source term
14	%	c: initial condition
15	%	Accd1, bccd1: matrices to impose boundary conditions using Lagrange
		multipliers
16	%	W,w: matrices for the time-integration scheme
17	%	dt: time-step
18	%	nstep: number of time steps
19	%	referenceElement: Information about the element of reference
20	%	M: mass matrix
21	%	C: convection matrix
22	%	K: stiffness matrix
23	%	tau: stabilization parameter
24	%	nodesDir0: nodes where the value of u is 0
25	%	nodesDir1: nodes where the value of u is 1
26		
27	%	Number of points used in the discretization
28	nţ	poin = size(X, 1);
29		
30	%	COMPUTATION OF MATRIX K2
31		
32	K2	2 = CreK2Mat (X,T,Conv,referenceElement);
33		
34	%	Integration matrix
35	[r	[m,m] = size(W);
36	Id	t = eye(n,m);
37	ta	uW = tau*W;
38	taı	uW_W = tauW'*W;
39	ta	$uw_w = tauw' *w;$





```
40
41 if size(tau) == [1,1]
42
       tau = tau * ones(size(W));
43 end
44
45 % Computation of the matrix necessary to obtain solution at each time-
      step: A du = F
46 disp('')
47 disp('Computation of total matrices for the time-integration scheme')
48 Kt = C + nu*K+sigma*M;
49 A = [];
50 for i = 1:n
      row = [];
51
       for j = 1:m
52
           row = [row, Id(i, j) * M + dt * W(i, j) * Kt + ...
53
54
                          tauW(j,i)*C' + dt*tauW_W(i,j)*(K2+sigma*C') ];
       end
55
       A = [A; row];
56
57 end
58
59 nccd = size (Accd1, 1);
60 Accd = []; bccd = [];
61 for i = 1:n
62
      row = [];
       for j = 1:m
63
           row = [row, Id(i, j) * Accd1];
64
65
       end
       Accd = [Accd; row];
66
67
       bccd = [bccd; bccd1];
68 end
69
70 nccd = n*nccd;
71 Atot = [A Accd'; Accd zeros(nccd)];
72
73 % Factorization of matrix Atot
74 disp('')
75 disp('Factorization of the matrices for the time-integration scheme')
76 [L,U] = lu(Atot);
77 L = sparse(L);
78 U = sparse(U);
79
80 % Initial condition
81 Sol = c;
82 % Loop to compute the transient solution
```





```
83 disp('')
84 disp('Transient analysis: computation of the solution at each time step'
       )
85 for i = 1:nstep
        aux1 = -dt*(Kt*c);
86
87
        aux2 = -dt * ((K2+sigma*C')*c);
        F = [];
88
            for j=1:m
89
                   F = [F; w(j) * aux1 + tauW_w(j) * aux2];
90
            end
91
       F = [F; bccd];
92
        dc = U \setminus (L \setminus F);
93
        dc = reshape(dc(1:n*npoin), npoin, n);
94
        for k =1:size(dc,1)
95
            if any(nodesDir1 == k);
96
97
                 c(k) = 1;
            elseif any(nodesDir0 == k);
98
                 c(k) = 0;
99
            else
100
                 c(k) = c(k) + sum(dc(k,:),2);
101
102
            end
        end
103
104
        Sol = [Sol c];
105 end
 1 function Sol = GLS(X, T, Conv, nu, sigma, f, c, Accd1, bccd1, W, w, dt, nstep,
       referenceElement, M, C, K, nodesDir0, nodesDir1, tau)
 2 % Sol = SUPG(X, IEN, Conv, nu, f, c, Accdl, bccdl, T, s, beta, dt, nstep, tau)
 3 % This function computes solution for a transient convection-diffusion
       equation
 4 % at different time instants.
 5 % GLS method is used to perform spatial discretization.
 6 %
 7 % Input:
       X: nodal coordinates
 8 %
 9 %
        T: connectivities
        Conv: velocity field
10 %
11 %
        nu: diffusion
12 %
        sigma: reaction term
13 %
        f: source term
        c: initial condition
14 %
15 %
       Accd1, bccd1: matrices to impose boundary conditions using Lagrange
       multipliers
```

16 % W,w: matrices for the time-integration scheme







```
17 %
       dt: time-step
       nstep: number of time steps
18 %
       referenceElement: Information about the element of reference
19 %
20 %
       M: mass matrix
       C: convection matrix
21 %
       K: stiffness matrix
22 %
       tau: stabilization parameter
23 %
24 %
       nodesDir0: nodes where the value of u is 0
       nodesDir1: nodes where the value of u is 1
25 %
26
27 % Number of points used in the discretization
28 npoin = size(X, 1);
29
30 % COMPUTATION OF MATRIX K2
31
32 K2 = CreK2Mat (X,T,Conv,referenceElement);
33
34 % Integration matrix
35 [n,m] = size(W);
36 \text{ Id} = eye(n,m);
37 \text{ tauW} = \text{tau} \ast W;
38 \text{ tauw} = \text{tau} \ast w;
39 tauW_W = tauW'*W;
40 tauW_w = tauW'*w;
41
42 if size(tau) == [1,1]
43
       tau = tau * ones(size(T));
44 end
45
46 % Computation of the matrix necessary to obtain solution at each time-
      step: A du = F
47 disp('')
48 disp('Computation of total matrices for the time-integration scheme')
49 Kt = C + nu*K+sigma*M;
50 A = [];
51 for i = 1:n
       row = [];
52
       for j = 1:m
53
            row = [row, Id(i,j)*M + dt*W(i,j)*Kt + ...
54
                tau(i, j) * (1/dt) * M + tauW(i, j) * (C+sigma * M) + ...
55
                tauW(j, i) * (C' + sigma *M) + dt * tauW_W(i, j) * (K2 + sigma *C') + \dots
56
                dt * tauW_W(i, j) * sigma * (C+sigma * M) ];
57
       end
58
        A = [A; row];
59
```





```
60 end
61
62 nccd = size(Accd1,1);
63 Accd = []; bccd = [];
64 for i = 1:n
65
       row = [];
        for j = 1:m
66
67
            row = [row, Id(i, j) * Accd1];
        end
68
       Accd = [Accd; row];
69
        bccd = [bccd; bccd1];
70
71 end
72
73 nccd = n*nccd;
74 Atot = [A Accd'; Accd zeros(nccd)];
75
76 % Factorization of matrix Atot
77 disp('')
78 disp('Factorization of the matrices for the time-integration scheme')
79 [L,U] = lu(Atot);
80 L = sparse(L);
81 U = sparse (U);
82
83 % Initial condition
84 \text{ Sol} = c:
85 % Loop to compute the transient solution
86 disp('')
87 disp('Transient analysis: computation of the solution at each time step'
       )
88 for i = 1:nstep
        aux1 = -dt * Kt * c;
89
        aux2 = -(C+sigma*M)*c;
90
        aux3 = dt * (-(K2+sigma * C' + sigma * (C + sigma * M)) * c);
91
       F = [];
92
            for j=1:n
93
                   F = [F; w(j)*aux1 + tauw(j)*aux2 + tauW_w(j)*aux3];
94
            end
95
       F = [F; bccd];
96
        dc = U \setminus (L \setminus F);
97
        dc = reshape(dc(1:n*npoin), npoin, n);
98
        for k =1:size(dc,1)
99
            if any(nodesDir1 == k);
100
                 c(k) = 1;
101
102
            elseif any(nodesDir0 == k);
```





```
103
                c(k) = 0;
            else
104
                 c(k) = c(k) + sum(dc(k,:),2);
105
            end
106
107
       end
108
        Sol = [Sol c];
109 end
 1 function [W,w, beta]=Pade(d)
 2
 3 % Compute the matrices needed to perform the time integration Pad scheme
 4
 5 if d == 0
       W = 1/2;
 6
 7
       w = 1;
       beta = [0,1];
 8
 9 elseif d == 1
       W = (1/24) * [7 -1; 13 5];
10
       w = [1/2; 1/2];
11
12
       beta = [0, 1/2, 1];
13 elseif d == 2
       W = [0 \ 0; \ 1 \ 0];
14
       w = [1/2; 1/2];
15
       beta = [0, 1/2, 1];
16
17 else
        error('Unavailable time integration scheme')
18
19 end
20
21 end
```





